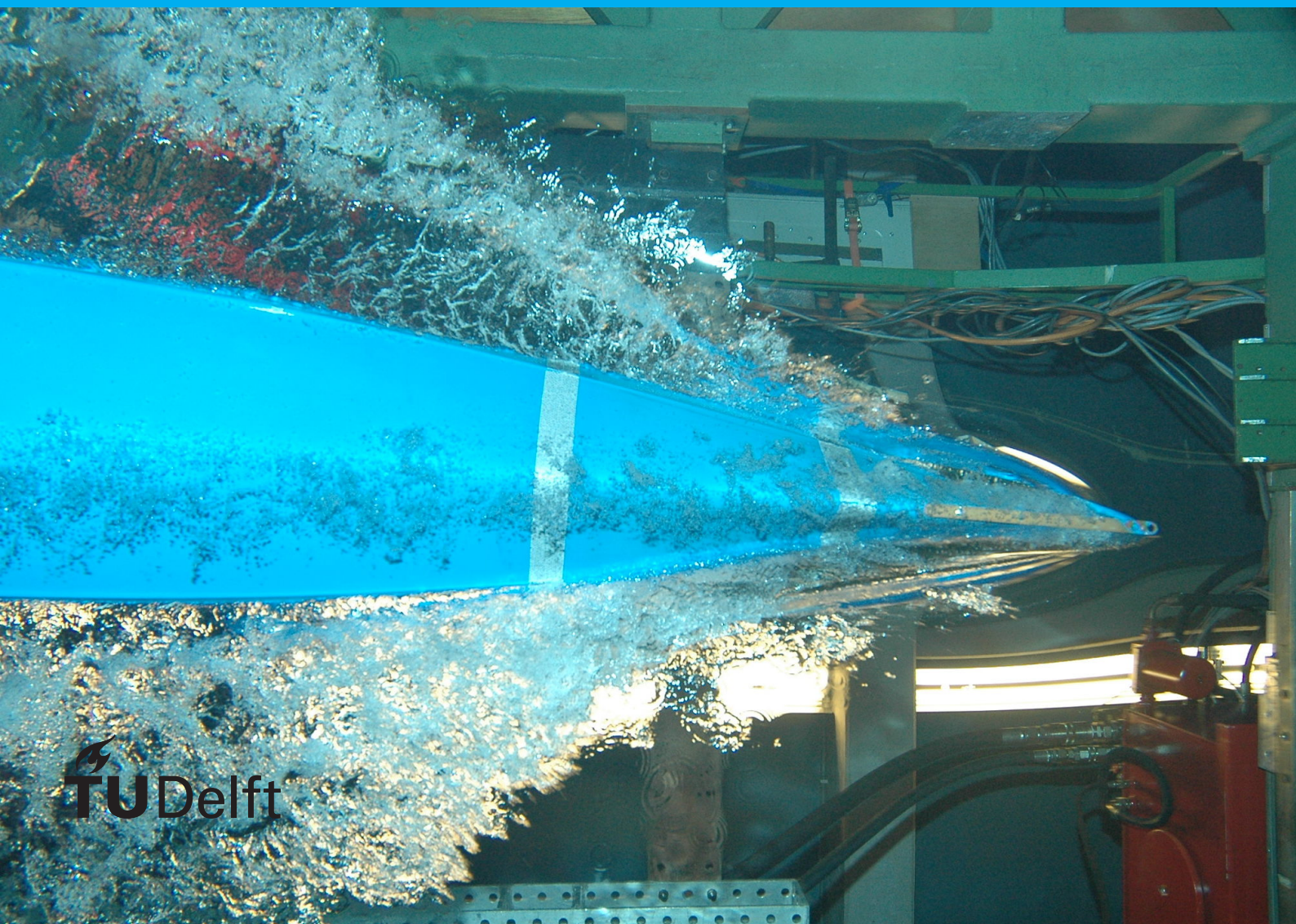
Mechanism Design for Optimal Contest

Y.J. Gu



Mechanism Design for Optimal Contest

Y.J. Gu

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	5353777	
Thesis committee:	Dr. R. J. Fokkink,	TU Delft, supervisor
	Prof. Dr. A. Papapantoleon,	TU Delft
	Dr. Ir. J. H. Weber,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Contents

Preface

This thesis is written as part of my master's in Applied Mathematics.

This academic journey has been a transformative experience, and I owe much of my growth to the invaluable guidance and support of my supervisor, Dr. Ir. R.J. Fokkink, whose tireless dedication to my academic development has been instrumental in shaping this work. From the beginning of this research to its completion, he has been a pillar of support, providing me with invaluable insights, scholarly wisdom, and encouragement.

Additionally, I extend my sincere gratitude to the thesis committee members for their invaluable contributions and commitment to evaluating my work-Dr.Ir.R.J.Fokkink, Prof.Dr.A.Papapantoleon, and Dr.Ir.J.H.Weber. I would like to extend heartfelt gratitude to my family and friends whose support has been a constant source of strength throughout this academic journey.

Y.J. Gu
Delft, December 2023

Abstract

In this thesis report, we delve into a contest game within game theory, where agents' risk-taking capacity, rather than effort, becomes the pivotal variable. High-quality performance in the game is associated with a higher probability of leading to superior scores through the use of risky strategies. The application of this contest game to labor markets prompts considerations for both employees and employers. From the perspective of employees, understanding the balance between risk-taking and payoff can be helpful in decision-making. On the employer's side, the efficiency of selection mechanisms becomes a critical factor.

We assess selection efficiency by examining the winning rates of high-type individuals. We use two parameters—market quality and market size in our analysis. Surprisingly, our theoretical analysis reveals a non-monotonic relationship between these factors and selection efficiency. Contrary to expectations, we find that as market quality improves or the number of agents increases, the winning rates of high types may decrease, resulting in reduced selection efficiency for employers.

Simulation experiments inspired by Fictitious Play and evolutionary game theory are conducted to research deeper into these dynamics. Learning rules and replicator dynamics under four scenarios are designed to address the inherent volatility in agents' strategic choices, test optimal strategies, and enable a comprehensive comparison of selection efficiency. A mechanism is proposed, derived from agents gaining experience from their usual behavior, and attempts to align outcomes more closely with Nash equilibrium, improving the optimal result. The study's unexpected findings about single-round screening in certain conditions highlight the need for tailored selection processes in different markets.

In summary, this research brings a fresh perspective to contest games. It encourages a rethink of traditional ideas and provides practical insights for decision-makers, especially in labor markets.

1

Introduction

The European job market is dynamic and continually evolving, influenced by various factors that impact both employees and employers. This thesis explores recent changes, addresses challenges faced by candidates and employers, and applies game theory to evaluate candidate selection methods. Through this analysis, the goal is to optimize selection processes and gain insights into the intricate dynamics at play.

The European employment market has experienced some shifts in recent years, propelled by global economic events and regional influences like the 2008 financial crisis and the unprecedented challenges from the COVID-19 pandemic. These events have reshaped the structural foundations of employment and prompted organizations to rethink traditional models. Eurostat data reveals an annual overview of the unemployment rate, represented by the percentage of unemployed individuals in the labor force. The labor force encompasses both employed and unemployed individuals aged 15 to 74. Notably, recent trends in Europe indicate that the unemployment rate has reached historically low levels.

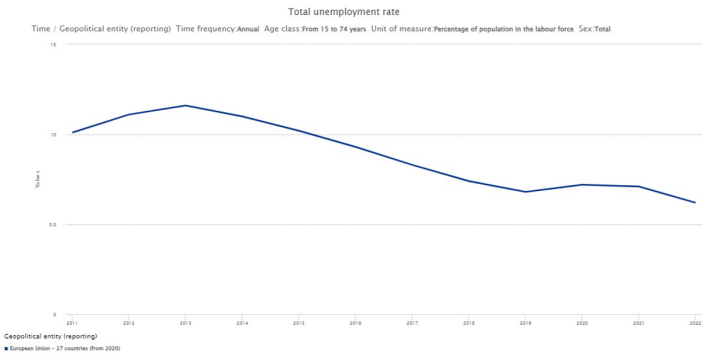


Figure 1.1: Unemployment rate in EU over years(Source: Eurostat, Statistics Netherlands)

In the Netherlands, the annual data for unemployment has dropped by more than half in the past ten years. Notably, the number of people who have been unemployed for a long time (more than one year) has decreased significantly.

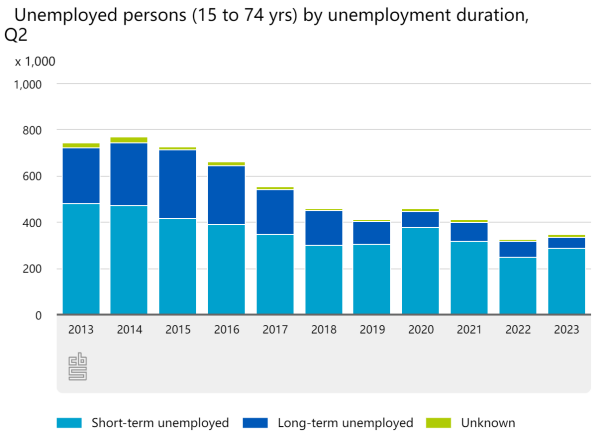


Figure 1.2: Unemployed person in NL over years

As candidates navigate the complex European job market, the focus has shifted from traditional career paths to emphasizing skills and adaptability. However, data from Statistics Netherlands (CBS) reveals an interesting trend in the educational backgrounds of the unemployed. In the second quarter of 2023, the average educational level of unemployed individuals has significantly risen compared to a decade ago.

Specifically, the statistics show that 31 percent of the unemployed had a high level of education (HBO, WO), marking an increase from 20 percent in 2013. Although the overall population also witnessed a rise in the proportion of highly educated individuals, the change was less pronounced, going from 28 percent in the second quarter of 2013 to 36 percent in the same period of 2023.

Both the long-term and short-term unemployed experienced a rise in the proportion of highly educated individuals. This increase was more pronounced among the long-term unemployed. The data for the second quarter of 2023 indicates that 29 percent of short-term unemployed individuals were highly educated, while among the long-term unemployed, this figure was higher at 40 percent. This trend suggests a noteworthy shift in the educational profiles of the unemployed in the Netherlands.

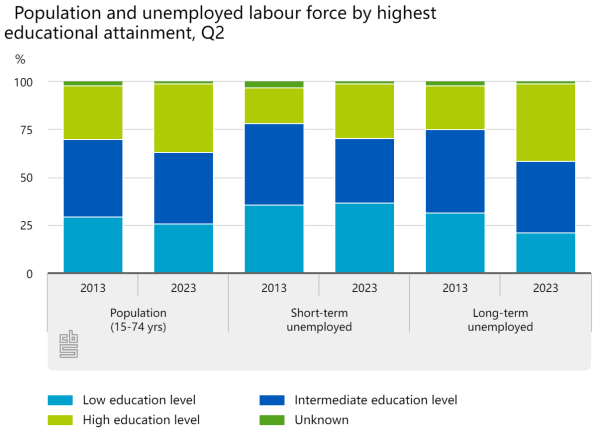


Figure 1.3: Unemployment rate by highest educational attainment in NL

At the same time, it is shown that up to 56% of new executives fail within the first two years of hire (Boydell et al., 2005), a success rate even lower than the probability of winning by flipping a coin. Hiring the best person for a job is a very difficult problem. In recent years, scholars within the field of game theory have increasingly delved into the intricate dynamics of candidate attributes. There is an urgent need to find a way to identify top candidates and make better use of labor quality.

In Chapter 1, we introduce various game theory definitions, laying the foundation for the subsequent analyses. In Chapter 2, we build upon the work of Hvide and Kristiansen, 2003. This chapter conducts a comprehensive mathematical analysis of a contest game. Solutions are derived, and the selection efficiency of the game is rigorously tested. In Chapter 3, simulations based on the analytical results are presented.

The scenarios considered include Agents without Memory, Agents with Memory, Multi-Agent scenarios, and Multi-Round contests. Each simulation is designed to further validate the theoretical analyses.

2

Mathematical Background

To figure out the complexity of competition games within the employment market, we start with the basics of game theory. We explore three games – the odd-even game, the secretary problem, and the Hawk-Dove game – to learn about strategic decision-making in different scenarios.

We begin with the odd-even game, employing direct computation and demonstrating how agents learn through fictitious play. Next, we introduce the secretary problem, a game where candidates are evaluated one after another instead of taking turns.

Recognizing the reality that real-world agents might not always make rational decisions, we delve into the concept of learning and adaptation. Agents start with initial strategies, refining them iteratively through a learning process. We use the Hawk-Dove game as a compelling example to illustrate these concepts.

2.1. Game theory

Game theory is the study of mathematical models of strategic interactions among rational agents. The discipline typically involves three fundamental elements:
1) Players: This refers to the set of participants involved in the game.
2) Actions: Actions represent the moves that each player can make during the game.
3) Payoffs: Payoffs are the scores or outcomes that each player receives at the end of the game.

Let's consider the most classical game in game theory, which is called a two-player zero-sum game. This is a game with only two players in which one player wins what the other player loses.

2.1.1. The Strategic Form

A "strategic form" is a formal and compact representation of a game that includes all the information needed to analyze the strategies and outcomes of the game.

Definition 1 (**Strategic Form of Two Zero-Sum Game**)**.** The strategic form of a two-person zero-sum game is given by a triplet (X, Y, A), where X, Y represents the set of strategies of Player I and II respectively, commonly referred to as pure strategies. A is a real-valued function defined on $X \times Y$. $A(x, y)$ represents the payoff function of player I, which means the winnings of player I and the losses of player II (Ferguson, 2020).

If Player I has m actions, $1, 2, \ldots, m$, Player II has n actions, $1, 2, \ldots, n$. The payoff matrix $A \in \mathbb{R}^{m \times n}$ represents the payoff to Player I:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

If Player I chooses i and Player II chooses j, the payoff to Player I is a_{ij} and the payoff to Player II is $-a_{ij}$. The sum of the payoff to Player I and the payoff to Player II is 0 (Karlin and Peres, 2017).

In the following part, we will use an odd-even game as an example to explain some basic concepts. An odd-even game is a zero-sum game involving two sides, resulting in an advantage for one side and an equivalent loss for the other. Two players take turns selecting numbers from a pool. After each selection, the players

add the numbers together. If the sum is an odd number, the player who correctly predicted the result (odd or even) earns a point.

Now consider an example of an odd-even game with $X = \{1,2\}$, $Y = \{1,2\}$, and A as given in the following table.

		Player II(even) 1	Player II(even) 2
Player I(odd)	1	$(-2,+2)$	$(+3,-3)$
	2	$(+3,-3)$	$(-4,+4)$

$$A(x,y) = \begin{pmatrix} -2 & +3 \\ +3 & -4 \end{pmatrix}$$

Definition 2 (**Mixed Strategies**)**.** While a pure strategy is an element of a strategy set, a mixed strategy is a probability distribution over the available pure strategies in the player's strategy set.

For any two-person zero-sum game with payoff matrix $A \in \mathbb{R}^{m\times n}$, a mixed strategy for Player I may be represented by a column vector, $(p_1, p_2, \ldots, p_m)^T$ of probabilities that add to 1. Similarly, a mixed strategy for Player II is an n-tuple $\boldsymbol{q} = (q_1, q_2, \ldots, q_n)^T$. The sets of mixed strategies of players I and II will be denoted respectively by X^* and Y^*,

$$X^* = \left\{ \boldsymbol{p} = (p_1, \ldots, p_m)^T : p_i \geq 0, \text{ for } i = 1, \ldots, m \text{ and } \sum_1^m p_i = 1 \right\}$$

$$Y^* = \left\{ \boldsymbol{q} = (q_1, \ldots, q_n)^T : q_j \geq 0, \text{ for } j = 1, \ldots, n \text{ and } \sum_1^n q_j = 1 \right\}$$

2.1.2. The Minimax Theorem

When Player II uses a mixed strategy $\boldsymbol{q} \in Y^*$, and Player I selects row i, the average payoff is given by $\sum_{j=1}^n a_{ij} q_j = (\boldsymbol{Aq})_i$. If Player I knows that Player II is going to use a particular mixed strategy $\boldsymbol{q}$, Player I would choose the row i between 1 and m or, in other words, $\boldsymbol{p} \in X^*$, to maximize the payoff:

$$\max_{1\leq i\leq m} \sum_{j=1}^n a_{ij} q_j = \max_{\boldsymbol{p}\in X^*} \boldsymbol{p}^T \boldsymbol{Aq}$$

Consequently, Player II is incentivized to choose $\boldsymbol{q}$ in a way that minimizes this payoff.

Similarly, in the reverse scenario where Player I makes the initial decision, Player II would choose column j between 1 and n to minimize the average payoff:

$$\min_{1\leq i\leq n} \sum_{i=1}^m p_i a_{ij} = \min_{\boldsymbol{q}\in Y^*} \boldsymbol{p}^T \boldsymbol{Aq}$$

In response, Player I would then select $\boldsymbol{p}$ to maximize this payoff. This leads to the concept of the upper and lower values of a game.

Definition 3 (**Upper and Lower Values of a Game**)**.** The upper value of the game (X, Y, A) is the minimum average loss that Player II can achieve no matter what Player I does, defined by:

$$\bar{V} = \min_{\boldsymbol{q}\in Y^*} \max_{1\leq i\leq m} \sum_{i=1}^n a_{ij} q_j = \min_{\boldsymbol{q}\in Y^*} \max_{\boldsymbol{p}\in X^*} \boldsymbol{p}^T \boldsymbol{Aq}.$$

The lower value of the game is the maximum payoff Player I can obtain, defined by:

$$\underline{V} = \max_{\boldsymbol{p}\in X^*} \min_{1\leq j\leq n} \sum_{i=1}^m p_i a_{ij} = \max_{\boldsymbol{p}\in X^*} \min_{\boldsymbol{q}\in Y^*} \boldsymbol{p}^T \boldsymbol{Aq}$$

Theorem 1 (**Minimax Theorem**)**.**

$$\bar{V} = \max_{\boldsymbol{p}\in X^*} \min_{\boldsymbol{q}\in Y^*} \boldsymbol{p}^T \boldsymbol{Aq} = \min_{\boldsymbol{q}\in Y^*} \max_{\boldsymbol{p}\in X^*} \boldsymbol{p}^T \boldsymbol{Aq} = \underline{V}.$$

Published in 1928, von Neumann's minimax theorem states that: There is a number V, called the value of the game, For Player I, by selecting a mixed strategy $\boldsymbol{p} \in X^*$, maximizing the minimum expected payoff, the average gain is at least V, no matter what II does. For Player II, by selecting a mixed strategy $\boldsymbol{q} \in Y^*$, minimizing the maximum expected loss, the average loss is at most V, no matter what Player I does. If V is zero we say the game is fair; if V is positive, we say the game favors Player I; if V is negative, we say the game favors Player II.

2.1.3. The Principle of Indifference

From the Minimax theorem, in a finite game, both players have optimal mixed strategies, $\boldsymbol{p}$ and $\boldsymbol{q}$. Player I will search for an optimal strategy in $\boldsymbol{p}$ that makes Player 2 indifferent as to which of the (good) pure strategies to use. And Player I's average payoff is at least V (Ferguson, 2020). Therefore we have:

$$\sum_{i=1}^{m} p_i a_{ij} \geq V \quad \text{for all } j = 1, \ldots, n.$$

Similarly, Player II's optimal strategy $\boldsymbol{q}$ also makes Player 1 indifferent among pure strategies in $\boldsymbol{q}$: $\boldsymbol{q} = (q_1, \ldots, q_n)^T$ is optimal for II if and only if

$$\sum_{j=1}^{n} a_{ij} q_j \leq V \quad \text{for all } i = 1, \ldots, m.$$

Definition 4 (**Equilibrium Theorem**)**.** When both players use their optimal strategies, $\boldsymbol{p} = (p_1, \ldots, p_n)^T$ and $\boldsymbol{q} = (q_1, \ldots, q_n)^T$ respectively, then

$$\sum_{j=1}^{n} a_{ij} q_j = V \quad \text{for all } i \text{ for which } p_i > 0$$

and

$$\sum_{i=1}^{m} p_i a_{ij} = V \quad \text{for all } j \text{ for which } q_j > 0$$

This means that if one player is playing optimally, any action that has positive weight in the other player's optimal mixed strategy is a suitable response. It implies that any mixture of these "active actions" is a suitable response.

To find the optimal strategies in the odd-even game mentioned before, we will first need to check the saddle point.

Definition 5 (**Saddle Point**)**.** If some entry a_{ij} of the matrix $\boldsymbol{A}$ has the property that (1) a_{ij} is the minimum of the i th row, and (2) a_{ij} is the maximum of the j th column, then we say a_{ij} is a saddle point. If a_{ij} is a saddle point, then Player I can then win at least a_{ij} by choosing row i, and Player II can keep her loss to at most a_{ij} by choosing column j. a_{ij} is the value of the game.

Recall the payoff matrix

$$A(x, y) = \begin{pmatrix} -2 & +3 \\ +3 & -4 \end{pmatrix}$$

There is no such a saddle point. Then we can compute the optimal mixed strategy. In our example, if Player I calls one with probability p_1 and calls two with probability p_2, we want to find them such that no matter what Player II chooses, player I has the same payoff. Therefore,

$$\begin{aligned} -2p_1 + 3(1 - p_2) = 3p_1 - 4(1 - p_2) &= V \\ p_1 + p_2 &= 1 \\ p_1 &= \frac{7}{12} \\ V = 3 \cdot \frac{7}{12} - 4 \cdot \frac{5}{12} &= \frac{1}{12} \\ 2q_1 - 3(1 - q_2) = -3q_! + 4(1 - q_2) &= V \\ q_1 + q_2 &= 0 \\ q_1 &= \frac{7}{12} \end{aligned}$$

The strategy for Player I is $\boldsymbol{p} = (7/12, 5/12)^T$, strategy for Player II is also $\boldsymbol{q} = (7/12, 5/12)^T$. Both players have the same optimal strategies because the matrix is symmetric. The value of the game is $\frac{1}{12}$, indicating an advantage for Player I since the value is positive.

2.1.4. Fictitious play

Fictitious play is a dynamic learning process where players iteratively update their strategies based on perceived opponent behavior. Introduced by George Brown(Brown, 1951), this method aims to converge to the value of a zero-sum game. The process unfolds sequentially, approximating the game's value with increasing precision. After playing n rounds, Player II, with a perfect memory of Player I's past choices, tries to predict Player I's strategy by averaging the n previous choices. After each round, both players update their understanding of the opponent's strategy. This iterative approach offers upper and lower bounds that converge toward the true value of the game, along with strategies for the players that achieve these bounds. The advantage of fictitious play is its simplicity and flexibility, allowing users to stop the process at any point and obtain answers known. The players interact in rounds as follows (Ferguson, 2020):
1) Player I chooses an arbitrary initial pure strategy $1 \le i_1 \le m$. Then j_1 is chosen as that j which minimizes $A(i_1, j)$. Similarly, if j_1 have already been chosen, i_2 is then chosen as that i that maximizes $A(i, j_1)$. To be specific, we define

$$s_k(j) = \sum_{\ell=1}^{k} A(i_\ell, j) \quad \text{and} \quad t_k(i) = \sum_{\ell=1}^{k} A(i, j_\ell)$$

2) After k rounds, we have sequences i and j of strategies in these rounds. We choose i_{k+1} as the smallest value of i that maximizes $t_k(i)$. Similarly, j_k is taken as the smallest j that minimizes $s_k(j)$. We define:

$$j_k = \operatorname{argmin} s_k(j) \quad \text{and} \quad i_{k+1} = \operatorname{argmax} t_k(i)$$

3) Notice that $\bar{V}_k = (1/k)\, t_k(i_{k+1})$ is an upper bound to the value of the game since Player II can use the strategy that chooses j randomly and equally likely from $j_1, \ldots, j_k$ and keep Player I's expected return to be at most $\bar{V}_k$. Similarly, $\underline{V}_k = (1/k) s_k(j_k)$ is a lower bound to the value of the game.

We use the same example to see how to use fictitious play. Recall that the matrix for the game is

$$A = \begin{pmatrix} -2 & +3 \\ +3 & -4 \end{pmatrix}$$

We take the initial $i_1 = 1$, and find

k	i_k	$s_k(1)$	$s_k(2)$	$\underline{V}_k$	j_k	$t_k(1)$	$t_k(2)$	$\bar{V}_k$
1	1	**−2**	3	−2	1	−2	**3**	3
2	2	1	**−1**	−0.5	2	**1**	−1	0.5
3	1	**−1**	2	−1/3	1	−1	**2**	2/3
4	2	2	**−2**	−0.5	2	**2**	−2	0.5
5	1	**0**	1	0	1	0	**1**	1/5
6	2	3	**−3**	−1/2	2	**3**	−3	0.5
7	1	1	**0**	0	2	**6**	−7	6/7
8	1	**−1**	3	−1/8	1	**4**	−4	0.5
9	1	**−3**	6	−1/3	1	**2**	−1	2/9
10	1	**−5**	9	−1/2	1	0	**2**	1/5
11	2	**−2**	5	−2/11	1	−2	**5**	5/11
12	2	**1**	1	1/12	1	−4	**8**	8/12
13	2	4	**−3**	−3/13	2	−1	**4**	4/13
14	2	7	**−7**	−1/2	2	**2**	0	1/7

The initiation of $i_1 = 1$ yields $(s_1(1), s_1(2), s_1(3))$ as the initial row of matrix A, with the minimum value residing at $s_1(1)$. Consequently, $j_1 = 1$. The second column of A designates $t_1(2)$ as the maximum, resulting in $i_2 = 2$. Subsequently, the sequences i_k and j_k are deterministically defined. The maximum of $\underline{V}_k$ discovered so far occurs at $k = 12$, with a value of $s_k(j_k)/k = 1/12$. Player I can secure this value by employing the mixed

strategy (7/12, 5/12), since in the first 12 instances of i_k, there are 7 occurrences of 1 and 5 occurrences of 2. The minimum of the $\bar{V}_k$ emerges multiple times, with a value of 1/7. Up to this analysis, the bounds for V are established as $1/12 \leq V \leq 1/7$. The player slowly converges to the Nash Equilibrium.

2.2. Optimal stopping and Secretary Problem

Assume there is a single available secretarial position. The total number of applicants is known. Applicants are interviewed one by one in a random order, with each possible order being equally likely. It is assumed that you have the ability to rank all the applicants from best to worst without any ties. Your decisions regarding accepting or rejecting an applicant are solely based on the relative rankings of those applicants interviewed up to that point. Once an applicant is rejected, they cannot be reconsidered later in the process. The interviewers have exceptionally high standards and will only be satisfied with the absolute best candidate. In other words, the interviewers' objective is to achieve a payoff of 1 by selecting the best candidate from the pool of n applicants, and the interviewer will receive a payoff of 0 otherwise. The primary goal is to select a strategy that maximizes the probability of identifying the best applicant, considering that applicants are presented in a randomly chosen order. The key challenge involves determining the optimal stopping point — the moment at which the evaluation of candidates should cease, and a selection should be made.

The game is presented in Thomas S. Ferguson's articlein 1989, and there's a simple solution to the question. Selecting integer "r" greater than 1, the rule involves the rejection of the first "r - 1" applicants and the subsequent selection of the next applicant who ranks highest in comparison to the previously observed applicants. Let i be the best applicant, $\phi_n(r)$ be the probability of selecting the best applicant.

$$\begin{aligned}\phi_n(r) &= \sum_{i=1}^{n} P(\text{ applicant } i \text{ is selected} \cap \text{ applicant } i \text{ is the best}) \\ &= \sum_{i=1}^{n} P(\text{ applicant } i \text{ is selected} \mid \text{ applicant } i \text{ is the best}) \cdot P(\text{ applicant } i \text{ is the best})\end{aligned}$$

When $i <= r-1$, the best applicant is rejected, thus the probability is 0 indeed. When $i >= r$, the chosen ith candidate is for sure the best candidate whenever the second-best one was rejected at the beginning.

$$\begin{aligned}\phi_n(r) &= \left[\sum_{i=1}^{r-1} 0 + \sum_{i=r}^{n} P\left(\begin{array}{l}\text{the best of the first } i-1 \text{ applicants} \\ \text{is in the first } r-1 \text{ applicants}\end{array} \mid \text{applicant } i \text{ is the best}\right)\right] \cdot \frac{1}{n} \\ &= \left[\sum_{i=r}^{n} \frac{r-1}{i-1}\right] \cdot \frac{1}{n} \\ &= \frac{r-1}{n} \sum_{i=r}^{n} \frac{1}{i-1}. \\ &= \frac{r-1}{n}\left(\sum_{i=1}^{n-1} \frac{1}{i} - \sum_{i=1}^{r-1} \frac{1}{i}\right)\end{aligned}$$

Since $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the $n-th$ harmonic number, $H_r = \sum_{i=1}^{r} \frac{1}{i}$ is the $r-th$ harmonic number, as 'n' tends toward infinity, H_n tends to $log(n) - \gamma$, where γ is the Euler constant. Therefore,

$$\phi_n(r) = \lim_{n\to\infty} \frac{r-1}{n}(H_{n-1} - H_{r-1}) \approx \lim_{n\to\infty} \frac{r-1}{n}(log(n) - \gamma - log(r-1) + -\gamma)$$

.

According to the concept of Euler constant, we have $\gamma = \lim_{n\to\infty}\left(-\log n + \sum_{i=1}^{n} \frac{1}{k}\right)$, therefore,

$$\phi_n(r) = \lim_{n\to\infty} (\log(n) - \gamma - (\log(r) - \gamma)) \cdot \frac{r}{n} \quad = \lim_{n\to\infty} (-\frac{r}{n} \log(\frac{r}{n}))$$

Represent x as the limit of $r - 1/n$.

$$\phi_n(r) \to -x\log(x)$$

Setting the derivative with respect to x equal to zero and then solving for x, which is $\frac{1}{e}$. The result shows that with increasing values of 'n,' in order to improve the selection efficiency, the optimal cutoff tends to approach 'n/e,' and the selection of the best applicant occurs with a probability of approximately '1/e.'

2.3. Evolutionary Game Theory

Evolutionary game theory is a framework, for understanding how strategic behaviors evolve in biological systems. It builds upon the foundations of game theory and expands our knowledge to go beyond just human decision-making and into the natural world. The key idea behind evolutionary game theory is its ability to model how individuals, in a population make choices as they adapt to their surroundings and interact with each other. Unlike classical game theory, it doesn't assume that players always act rationally. Instead, it merely requires individuals to possess a strategy, with the primary objective being the assessment of the effectiveness of that strategy.

The evolutionary game is non zero-sum, which is more difficult than a zero-sum game. The main purpose is to find the Nash Equilibrium.

Definition 6 (**Pure Nash Equilibrium**)**.** For pure strategy choices $X^* = \left(x_1^*, \ldots, x_i^*, \ldots, x_n^*\right)$ where $i = 1, 2, \ldots, n$, $u_i\left(x_1, x_2, \ldots, x_n\right)$ represents the payoff to player i. If we have

$$u_i\left(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n\right) \geq u_i\left(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n\right)$$

the pure strategies choice is a Nash equilibrium.

The definition can be extended while the players use mixed strategies.

Definition 7 (**Mixed Nash Equilibrium**)**.** Let X_i^* be the set of mixed strategies of player i. A vector of mixed strategy choices $\left(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\right)$ with $\boldsymbol{p}_i \in X_i^*$ for $i = 1, \ldots, n$ is said to be a strategic equilibrium, if for all $i = 1, 2, \ldots, n$, and for all $\boldsymbol{p} \in X_i^*$, the average payoff to player i

$$g_i\left(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{i-1}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \ldots, \boldsymbol{p}_n\right) \geq g_i\left(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{i-1}, \boldsymbol{p}, \boldsymbol{p}_{i+1}, \ldots, \boldsymbol{p}_n\right).$$

A Nash equilibrium is defined as a profile of strategies, implying an assignment of strategies to each player. It constitutes a mutual best response scenario, where no player possesses an incentive to deviate from their chosen strategy. If one player decides to deviate but all others do not, then this hurts the deviating player. This is sometimes called "one shot deviation".

However, in the evolutionary game, it is crucial to note that a Nash equilibrium falls short in ensuring evolutionary stability. A strategy profile x is a Nash Equilibrium (NE) if no player can improve their outcome by deviating. For another strategy profile y, where $x_i = y_i$ for all players except j, the condition $u_j(y, x) \leq u_j(x, x)$ holds. In Nash equilibrium, there are cases when $u_j(y, x) = u_j(x, x)$, indicating that player j might consider choosing y instead of x. An Evolutionarily Stable Strategy (ESS) has a stronger condition, where for other players, $u_i(y, x) > u_i(y, y)$ must hold. x is a best reply to itself but a strict best reply to y, which makes it an ESS. Here's an example of it. Consider the payoff matrix for Harm thy neighbor game as follows:

		Player II	
		A	B
Player I	A	$(2,2)$	$(1,2)$
	B	$(2,1)$	$(2,2)$

It is obvious that both (A, A) and (B, B) are Nash equilibrium. However, $u_1(B, B) > u_1(A, B)$ while $u_1(B, A) = u_1(A, A)$, which makes strategy B the only ESS.

2.3.1. Hawk-Dove Game

We can use the Hawk-Dove game to explain Evolutionary Game Theory. The hawk-dove game was first analyzed in the article 'The Logic of Animal Conflict'(Gadagkar, 2005). This model revolves around conflict between two players competing for a fixed resource. Each participant adopts one of two distinct strategies, as outlined below: Hawk: This strategy involves initiating aggressive behavior and persisting until injured or until the opponent yields. Dove: In contrast, the Dove strategy dictates an immediate retreat if the opponent initiates aggressive behavior. In interactions between players of the same type, Doves share resources equally, while Hawks, engaging in aggressive attacks, pose an equal probability of injury to both parties. Conflict reduces the fitness of the injured participant by a constant value denoted as C, and B is the value of the contested resource. When a Dove encounters a Hawk, the Hawk claims the resources. The traditional payoff matrix is shown as follows:

	Hawk	Dove
Hawk	(B − C)/2, (B − C)/2	B, 0
Dove	0, B	B/2, B/2

If $(B-C)>0$, there's only one pure Nash equilibrium $((B-C)/2,(B-C)/2)$ since Hawk strictly dominates Dove for both players. Note that if $(B-C)/2<0$, there're two pure Nash equilibriums (Hawk, Dove) and (Dove, Hawk) with payoff $(B,0)$ and $(0,B)$ respectively.

Now we simulate the game with an example:

	Hawk	Dove
Hawk	−1, −1	10, 0
Dove	0, 10	5, 5

Through mathematical analysis, the two Pure Strategy Equilibria (PSE), namely $(Dove, Hawk)$ and $(Hawk, Dove)$, are easily identified. Additionally, a mixed strategy is derived for Player I, involving the probability of choosing a Hawk move, denoted as p, while the probability of choosing a Dove move is $1-p$. Player II should be indifferent between the Hawk and Dove moves. Therefore, the following equation is established:

$$-p+10(1-p)=5(1-p)$$

Therefore, $p=5/6$. According to the definition of Evolutionary Stable Strategy (ESS), this mixed strategy, specifically $(\frac{5}{6},\frac{1}{6})$, can be also proven the ESS through the following calculations:

$$\begin{aligned} V(5/6,q) &= -\frac{5}{6}+10\cdot\frac{5}{6}*(1-q)+0\cdot\frac{1}{6}+5\cdot\frac{1}{6}*(1-q) \\ V(q,q) &= -q^2+10q(1-q)+5(1-q)^2 \\ V(5/6,q)-V(q,q) &= (6q-5)^2/6>0 \end{aligned} \tag{2.1}$$

This inequality holds as long as $q\neq\frac{5}{6}$, establishing the mixed strategy as evolutionarily stable with a population of 5/6 Hawks and 1/6 Doves.

In all, there are three Nash equilibriums. What is the NE that a population of birds will end up in? To empirically validate this result, several simulations are conducted. Beginning with a population of 1000 birds (999 Doves and 1 Hawk), the birds are paired into 500 pairs. Following the game rules, we can calculate the fitness of each bird after the first iteration.

Considering finite population random effects, we first experiment with the effect of the mutation. In case of a draw, the birds remain unchanged. If not, the losing Dove transforms into a Hawk. The simulation demonstrates a convergence towards an all-Hawk population, typically occurring around the 10th iteration. Following this learning rule, doves were eliminated, because the payoff for hawks was $(B–C)/2<0$, and the population went to extinction.

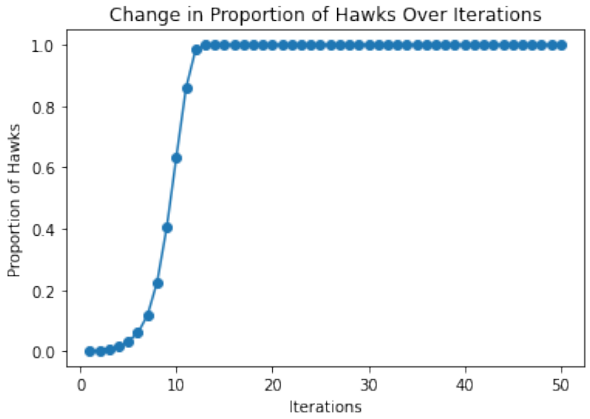


Figure 2.1: Hawk-dove(mutation)

To enhance the learning rule and avoid the extinction of doves due to direct mutations, we assume that there is only a small chance of mutation that may alter the type of offspring. This mutation only has a limited

effect, as the next generation is ultimately determined based on weight. Assuming the total number of birds remains constant while the composition changes, we select the next generation by giving more weight to birds with higher fitness. This increased weight enhances their chances of producing offspring. The result shows that the population ends up in the mixed Nash Equilibrium, which is aligned with the Evolutionary Stable Strategy as analyzed before.

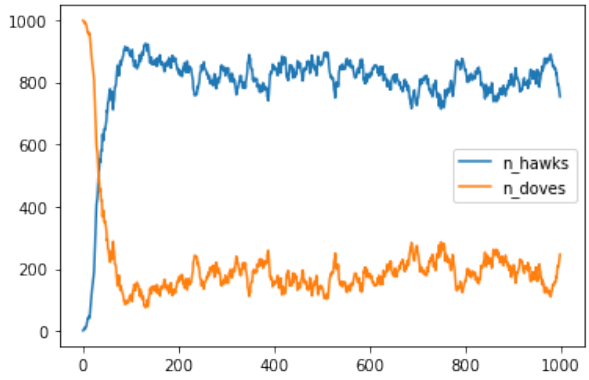


Figure 2.2: Hawk-dove game

2.4. Summary

In this chapter, we reviewed classical concepts from game theory, optimal selection, and evolutionary dynamics. These concepts lay the foundation for our analysis of contests in the job market, where applicants exhibit behavior like hawks and doves learning from each other. Additionally, the firms engaging with these applicants face optimal selection challenges. Our exploration of these mathematical background sets the stage for a deeper dive into a specific game theory paper in the next chapter.

3

Risk Taking in Selection Contests

In this chapter, we review the work of Hvide and Kristiansen, 2003, focusing on a contest game and deriving its solutions. The chapter commences with a definition of the game, followed by a comprehensive mathematical analysis of scenarios involving two and three agents, respectively. The primary focus of the paper is on the selection efficiency of contests, considering the number of contestants and the quality of the contestant pool-the proportion of high-type agents in the pool. Contrary to the intuitive expectation, the study reveals that an increase in the number of contestants or an improvement in the quality of the market does not consistently improve the selection efficiency. The selection efficiency may experience a decline in higher-quality markets. These findings are supported by an exploration of the contest game and the mathematical analysis.

3.1. Definition of the game

The secretary problem outlined in Chapter 2 is a renowned example of situations where applicants are presented sequentially, one after another, and the objective is to identify the optimal candidate with only limited information. In this section, we conduct an in-depth review of the model proposed by Hvide and Kristiansen, 2003. In their paper, the investigation revolves around the selection efficiency of Multi contests wherein contestants optimize their risk-taking choices considering the risk-taking behaviors of others. The study further delves into determining which type of contestant is likely to win and evaluates the factors influencing the efficiency of contest selection. Two pivotal factors are emphasized in the paper: the number of contestants and the quality of contestants.

In contrast to scenarios like the secretary problem, which primarily revolves around identifying the optimal strategy to maximize the probability of selecting the best applicant over rounds, the analysis undertaken by Hvide and Kristiansen introduces a unique perspective. Here, all applicants apply simultaneously, altering the dynamics as the decision-maker must concurrently evaluate multiple applicants and devise a strategy to choose the most suitable one. This deviation introduces a different set of complexities and considerations in the decision-making process.

The paper starts by considering two distinct types of agents: high types and low types, which are categorized within the type space denoted as $T = l, h$. The parameter θ represents the proportion of high-type agents within the pool from which a total of n agents are drawn. Furthermore, it defines an action space denoted as C, which encompasses the possible actions that each agent can undertake, including a safe move or a risky move, i.e., $C = \{\text{s}, \text{r}\}$.

Next, Hvide and Kristiansen conduct an in-depth examination of the game denoted as $\Gamma(n, \theta)$. Defining n as the number of applicants that are drawn at random from the pool of job seekers, the analysis is specifically tailored for the case when $n = 2$. In this game, each agent possesses information about their type as well as the values of n and θ. However, they remain uninformed about the types of the other contestants, rendering the game an incomplete information game. The rules are straightforward: n contestants simultaneously engage in one game, and the one with the highest score emerges as the winner.

We define a mapping f from T to C that gives all possibilities of pure strategies if the number of applicants is equal to $n = 2$. The mapping f from T to C has 4 possibilities: $f(l) = f(h) = s, f(l) = s; f(h) = r; f(l) = r; f(h) = s; f(l) = f(h) = r$. This set of strategies can be written as $S := \{(s,s),(s,r),(r,s),(r,r)\}$, with the first coordinate representing the value of f for the low type and the second coordinate representing the value of f

for the high type.

An applicant achieves a score and the applicant with the highest score is selected. Assume the score of agents is an individual output space Z consisting of four elements, $Z := \{z_1, z_2, z_3, z_4\}$, where $z_1 < z_2 < z_3 < z_4$. Additionally, the contestant game incorporates the following assumptions: if a low type agent chooses the safe move s, their output is guaranteed to be z_2. Conversely, if a high type agent chooses s, their output is guaranteed to be z_3. On the other hand, if a low-type agent opts for the risky move r, their output is z_1 with a probability of $1-x$, and z_4 with a probability of x. Similarly, if a high type agent plays r, their output is z_1 with a probability of $1-y$, and z_4 with a probability of y, where y exceeds x. For all the contestants, the one who has the highest output will win the contest.

Furthermore, the paper investigated the impact of contestant quality on the selection efficiency of a contest. The selection efficiency is quantified by the probability of a high-type agent winning the prize in the best response equilibrium (BNE), denoted as $\Pi(\Gamma)$.

Now we can draw the game tree and play this game of two players as follows. Firstly we discuss the situation where Player 1 is high type, followed by the case that Player 1 is low type.

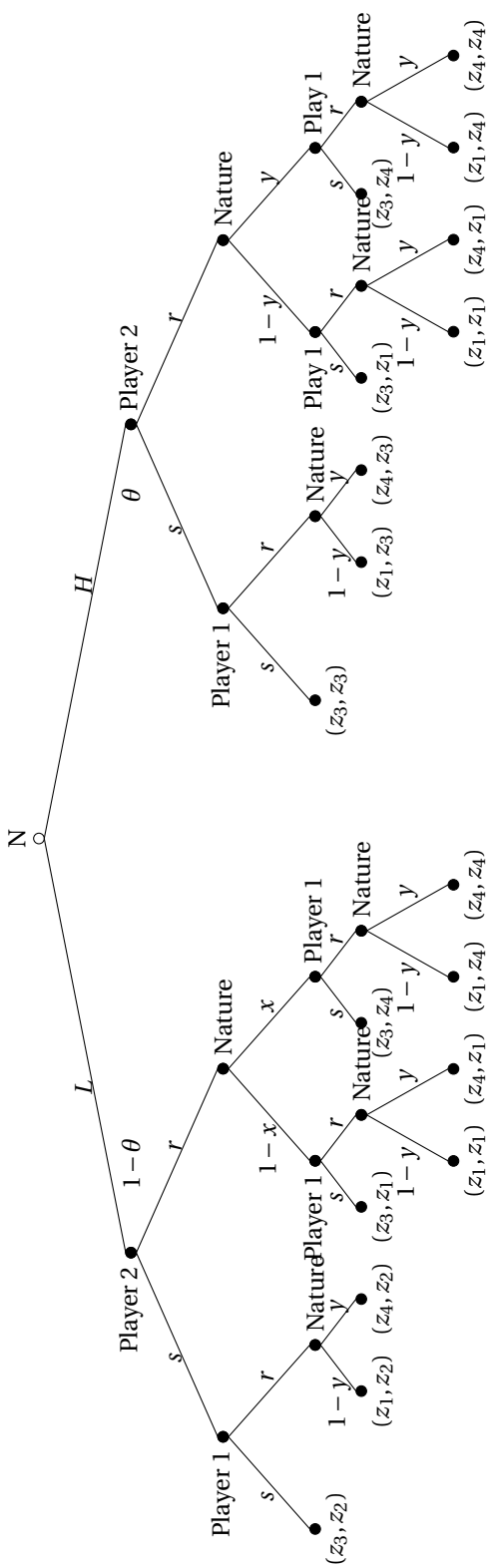


Figure 3.1: Game tree for high-type agents

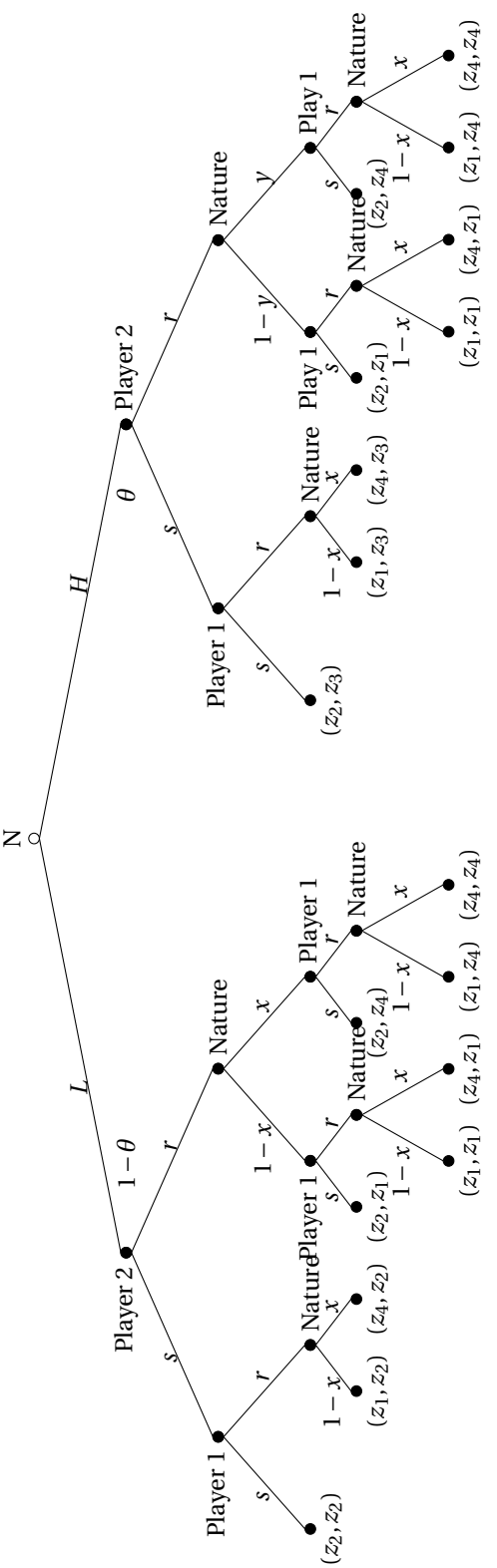


Figure 3.2: Game tree for low-type agents

3.2. Mathematical analysis

Having defined the game in the last section, we will now consider the optimal strategies of the players in the game. Particularly, we will only consider pure Nash Equilibrium but not mixed NE's. Recall that the player with the highest output wins. Let $U_i(j,k)$ represent the win probability of an agent of type i when agents of the same type (including themselves) play strategy j and agents of the other type play strategy k. We will now calculate the win probabilities for both types of agents under the four strategies.

In the scenario where $n = 2$, we first assume that player 1 is high-type. Considering this as an illustrative example, we proceed to calculate the win probability of the high-type agent when both types of agents opt for the safe strategy. For the strategy (s,s), wherein both players choose strategy s, we can observe that if the opponent is also a high type, the win probability for player 1 is $1/2$. However, if the opponent is a low type, the win probability for player 1 is 1. This analysis leads to the computation of $U_H(s,s)$ (the win probability of high types under strategy (s,s)) as follows:

$$U_H(s,s) = \frac{1}{2}\theta + (1-\theta) = 1 - \theta/2$$

Next, consider the scenario where the high-type agent chooses the safe strategy s and the low-type agent chooses the risky strategy r. When player 2 is a high type, the win probability for player 1 is $1/2$. However, when player 2 is a low type, player 1 wins only if player 2's output is z_1. Therefore, we have:

$$U_H(s,r) = \frac{1}{2}\theta + (1-\theta)(1-x) = 1 - x - \theta/2 + \theta x$$

Similarly, we can compute the win probabilities for player 1 under the pure strategies (r,r) and (r,s):

$$U_H(r,r) = \theta/2 + (1-\theta)[\frac{1}{2}(1-y)(1-x) + \frac{1}{2}xy + y(1-x)] = 1/2 + (1-\theta)(y-x)/2$$

$$U_H(r,s) = \frac{1}{2}\theta + (1-\theta)y$$

Now, let's assume player 1 is a low type. Following the same approach, we calculate the win probabilities for player 1 under the four pure strategies:

$$U_L(s,s) = \frac{1}{2}(1-\theta)$$

$$U_L(r,r) = \frac{1}{2}(1-\theta) + \frac{1}{2}\theta(1-x)(1-y) + \theta x(1-y) + \frac{1}{2}\theta xy = \frac{1}{2} + \frac{1}{2}\theta(x-y)$$

$$U_L(s,r) = \frac{1}{2}(1-\theta) + \theta(1-y)$$

$$U_L(r,s) = \frac{1}{2}(1-\theta) + \theta x$$

Using the one-shot deviation principle, we will analyze the potential strategy deviations to determine whether each strategy qualifies as a pure strategy equilibrium. By considering how agents might change their strategies to maximize their win probabilities, we can ascertain the stability of each strategy in the contest setting.

In the following analysis, we write $U_i'(j,k)$ to represent the win probability of an agent of type i when they play strategy $-j$ (the other strategy), while other agents of the same type opt for strategy j, and agents of the other type choose strategy k. Here's an example for $U_H'(r,r)$. Suppose player 1 is a high-type agent who chooses the safe strategy, while other high-type agents opt for the risky strategy, and the low-type agent chooses the risky strategy. In this scenario, the win probability for player 1 is as follows: if player 2 is also a high type, the win probability for player 1 is $\theta(1-y)$, corresponding to the case where the output for player 1 is $z3$ and for player 2 is $z1$. Conversely, if player 2 is a low type, the win probability for player 1 is $(1-\theta)(1-x)$, reflecting the case where the output for the low type is z_1. According to analysis, we have

$$U_H'(r,r) = \theta(1-y) + (1-\theta)(1-x)$$

Consider another scenario, specifically for $U_L'(r,s)$, where player 1 is a low-type agent select strategy s, making it different from other agents of the same type, who will choose strategy r. At the same time, the high-type agent will certainly choose the safe strategy. According to the analysis, we can compute:

$$U_L'(r,s) = (1-\theta)(1-x)$$

considering utility function $U'_H(s,r)$, the scenario involves a specific high-type agent opting for a risky strategy, while the remaining high types adhere to the safe strategy. All low types consistently select the risky strategy. The potential outcomes for the specific high-type agent to secure a victory are as follows: 1) Competing against other high-types and achieving a score of z_4, 2) encountering low types, with both parties obtaining a score of z_1 or z_4. Therefore, we have:

$$U'_H(s,r) = \theta y + (1-\theta)\left(\frac{1}{2}xy + y(1-x) + \frac{1}{2}(1-x)(1-y)\right)$$

Continuing with the same analytical approach, we proceed to calculate the win probabilities for player 1 of different types under various pure strategies.

$$\begin{aligned}
U'_H(r,s) &= \theta(1-y) + (1-\theta) \\
U'_H(s,s) &= y \\
U'_L(r,r) &= \theta(1-y) + (1-\theta)(1-x) \\
U'_L(s,r) &= \theta\left(\frac{1}{2}xy + x(1-y) + \frac{1}{2}(1-x)(1-y)\right) + (1-\theta)x \\
U'_L(s,s) &= x
\end{aligned}$$

Continuing from the previous analysis, we will now consider the Nash equilibrium with pure strategies, and find out how the parameters (x, y, θ) influence the choice of risk-taking.

3.2.1. Nash Equilibrium (r,r)

Starting with the strategy (r,r), we will check whether both types of agents adhere to the supposed equilibrium strategy. Specifically, we want to determine if $U_H(r,r) > U'_H(r,r)$ and $U_L(r,r) > U'_L(r,r)$ hold, where $U'(r,r)$ means that player deviates from r and plays s. If $U(r,r) > U'(r,r)$ then the player has no incentive to deviate. Referring to the previous equations, we have:

$$1/2 + (1-\theta)(y-x)/2 > \theta(1-y) + (1-\theta)(1-x)$$

Simplifying the inequality, we find that $y > [1-(2-\theta)x]/(2+\theta)$. Additionally, we have

$$\frac{1}{2} + \frac{1}{2}\theta(x-y) > \theta(1-y) + (1-\theta)(1-x)$$

thus $y > [1-(2-\theta)x]/\theta$. This condition is more restrictive than the previous one.

Simultaneously, we can analyze how the quality of the market influences the choice of risk-taking by examining the value of θ. It is obvious that if there are too many low agents in the market, leading to θ being quite small, as it appears in the denominator, y would become extremely large and unattainable. In such a case (r,r) will never be achievable, and the agents are inclined to the safe strategy. Expanding on the previously mentioned inequality, the attainment of the strategy (r,r) as a Nash equilibrium only materializes when θ surpasses a specific threshold, precisely when $\theta > \frac{1-2x}{y-x}$. At the same time, the bound on θ depends on the difference between y and x.In our model, we always have $y > x$, which means that high-type players perform better at the risky strategy. If y is close to x, then a high-type player doesn't have the advantage anymore and will prefer a safe move.

3.2.2. Nash Equilibrium (s,s)

Let $U_H(s,s) > U'_H(s,s)$ and $U_L(s,s) > U'_L(s,s)$, we have Nash equilibrium (s,s) if $y < 1-\theta/2$ for high type agent, and $x < \frac{1}{2}(1-\theta)$ for low type agent.

As for the influence of the proportion of high-type agents on risk-taking choices, it is observed that when $\theta < \min(2-2y, 1-2x)$, the strategy (s,s) is an equilibrium, where both types of agents opt for the safe action. However, as the proportion of high-type agents (θ) increases slightly and falls within the range as $\theta > \min(1-2x, 2-2y)$, a shift in the equilibrium is noted, under these conditions, there is an augmented probability of different types of agents choosing the risky strategy.

3.2.3. Nash Equilibrium (r,s)

Concerning the strategy (r,s), where low types consistently opt for the risky strategy and high types choose the safe strategy, for high-type agents, $U_H(r,s) > U'_H(r,s)$, the inequality is given by:

$$1 - x - \theta/2 + \theta x > \theta y + (1-\theta)\left(\frac{1}{2}xy + y(1-x) + \frac{1}{2}(1-x)(1-y)\right)$$

This simplifies to $y < \frac{1-(1-\theta)x}{1+\theta}$ and for low types,

$$\frac{1}{2}(1-\theta)+\theta x > (1-\theta)(1-x)$$

leads to $x > (1-\theta)/2$. These two inequalities combine to

$$1-2x < \theta < \frac{1-y-x}{y-x}$$

The low-type agents transition to the risky strategy, while the high-type agents adhere to the safe strategy. This choice is reasonable in situations with a higher proportion of high agents in the market, there is an increased probability of low-type agents encountering high types. In such a situation, the optimal strategy for low types to win is by producing output z_4 prompting their choice of the risky strategy.

3.2.4. Nash Equilibrium (s,r)

Finally considering strategy (s,r), following a similar analysis as before, we have

$$\frac{1}{2}\theta+(1-\theta)y > \theta(1-y)+(1-\theta)$$

which implies $y > 1-\theta/2$ And:

$$\frac{1}{2}(1-\theta)+\theta(1-y) > \theta\left(\frac{1}{2}xy+x(1-y)+\frac{1}{2}(1-x)(1-y)\right)+(1-\theta)x$$

Thus, $x < (1-\theta y)/(2-\theta)$ Additionally, we consider how variations in the quality of the market will influence this strategy. Certainly, when $2-2y < \theta < \frac{1-2x}{y-x}$ replaces $\theta < 2-2y$. the high-type agents shift to the risky strategy while the low-type agents maintain the safe strategy.

The analysis proves that the change in the equilibrium strategies is related to the variations in the proportion of high-type agents (θ). The inequality equations presented above also illuminate the correlation between the selected strategies and the parameters x, and y. For example, when x or y are relatively high, the agents are more inclined to opt for a risky strategy. The mathematical inequalities reach the same result: considering that θ falls within the range $0 < \theta < 1$, combined with $\theta < 1-2x$, for strategy (s,s), it shows that $0 < x < \frac{1}{2}$. This means that if x is larger than $1/2$, the low-type agents are supposed to choose the risky strategy for sure, and a safe move will never be reachable. For strategy (r,r), we have condition $\frac{1-2x}{y-x} < 1$, i.e. $1-x < y$. This means y is bigger than $1/2$, or high-type agents have no incentive to choose the risky strategy.

To visualize this result, we will use graphs to demonstrate the dynamics of the equilibrium strategies with changing range (x,y,z) When θ, x and y vary within the conditions: $0 < \theta < 1$, $0 < x < 1/2$ and $x < y < 1$.

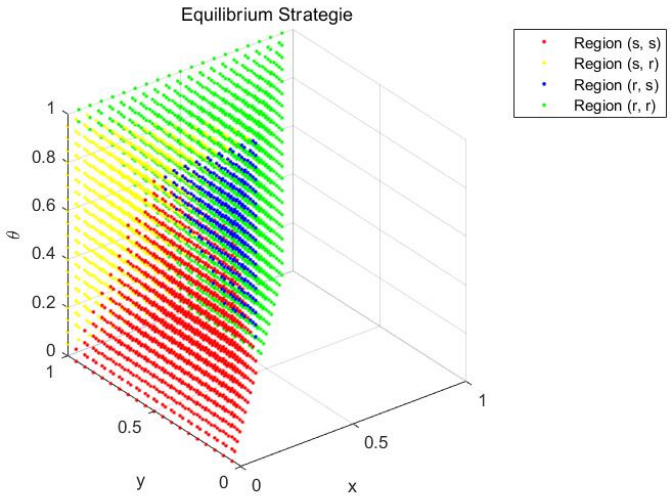


Figure 3.3: Nash equilibrium

The x-axis is the value of parameter x, the y-axis is the value of parameter y, and the z-axis corresponds to the parameter θ. The gradient of colors, transitioning from red to blue (or yellow) to green, represents the strategies from (s,s) to (r,s) or (s,r), and eventually to (r,r). The picture illustrates a significant trend: as the values of x, y, or θ increase, the game's equilibrium shifts towards a greater propensity for risk-taking. This finding aligns precisely with the results obtained from our computations.

3.3. Selection Efficiency

The main focus of the paper by Hvide and Kristiansen is the selection efficiency, denoted as $\Pi(\Gamma)$. This parameter represents the probability that a high-type contestant wins in the game, essentially securing a job offer. The paper uncovers an unexpected finding: as the percentage of high-types (denoted as θ) increases, the selection efficiency $\Pi(\Gamma)$ can decrease. In situations where candidates are hired randomly without a selection process, having more high types generally boosts efficiency because then the probability of a high type is θ. However, when a selection procedure is introduced, this is not the case. Surprisingly, having a higher proportion of high types doesn't always achieve higher selection efficiency.

For equilibrium strategy (s,s), low-type agents are only selected if both agents are low-type. This happens with probability $(1-\theta)^2$. Therefore, the probability of high types win is that:

$$\Pi(s,s) = 1 - P(low-type-wins) = 1-(1-\theta)^2 = 2\theta - \theta^2$$

In the case of strategy (s,r), low types are only successful if both agents are low types or if the high types obtain output z_1. The probability for high types to win is then:

$$\Pi(s,r) = 1 - P(low-type-wins) = 1-[(1-\theta)^2 + 2*(1-y)(1-\theta)\theta] = \theta^2 + 2*\theta(1-\theta)y$$

Notably, this is smaller than θ if $y < 1/2$. In this case, it is better for the employer to just select a candidate at random without a selection procedure. Regarding strategy (r,s), the only possibility for high types to win is to compete with high types or to obtain z_4 when competing with low types. We have:

$$\Pi(r,s) = P(high-type-wins) = \theta^2 + \theta(1-\theta)(1-x)$$

which is larger than θ, therefore the selection procedure is useful. For strategy (r,r), the situation for high types to win include:1) competing with high types, 2) competing with low types, and obtaining z_4, 3) competing with low types, and obtaining z_1 when low types also get $z1$:

$$\Pi(r,r) = \theta^2 + 2*(1-\theta)\theta\left[\frac{1}{2}xy + y(1-x) + \frac{1}{2}(1-x)(1-y)\right]$$

which is larger than θ, therefore the selection here is useful. It is evident that if the equilibrium is fixed, the function Π increases with the value of θ, as indicated by its positive first derivative. This finding demonstrates that increasing the quality of the contestant pool has a statistically significant effect, leading to a higher selection efficiency (Π). However, it may happen when one of the agents changes strategy if theta increases and that may lead to a worse Π. For example, $\Pi(s,r)$ and $\Pi(r,s)$ are lower than $\Pi(s,s)$. Consequently, Π may decrease as θ falls within the range $\min(1-2x, 2-2y) < \theta < \frac{1-2x}{y-x}$. However, when $\theta > \frac{1-2x}{y-x}$, the selection efficiency ($\Pi(r,r)$) returns to exhibiting a positive correlation with θ.

In summary, the study shows that the selection efficiency (Π) is positively influenced by an increase in the quality of the contestant pool (as θ increases), while also noting that an increase in contestant quality leads to a shift in equilibrium towards increased risk-taking, which may decrease selection efficiency eventually. To further illustrate this point, we provide two examples where we examine the impact of varying θ within the range from zero to one. For example 1, we set $x = 1/3$ and $y = 4/5$. For example 2, we have $x = 1/5$ and $y = 1/4$. By examining these two scenarios, we can observe how changes in θ influence the equilibrium strategies and, consequently, the selection efficiency of the contest.

To begin, the variable ν signifies the change in equilibrium as θ undergoes increments of 0.001, progressing from 0 to 1. In example 1, despite an overall increasing trend of Pi over θ, there are three transition points: when θ approaches 0.3, 0.4, the Nash equilibrium transitions from (s,s) to (r,s), then (s,r). This transition is accompanied by a decrease in Π and also indicates a shift towards a more risk-taking strategy. in medium-quality pool. When around θ of 0.7, there's the second transition point. The strategy changes from (s,r) to (r,r), where maintaining the strategy (s,r) would result in a larger Π. Afterward, Π exhibits a consistent increasing trend. In example 2, when both y and x are less than $1/2$, there exists a single transition point. From the

figure, it is evident that it is evident that in the beginning there's an increase, then when the transitions from (s,s) to (r,s), it results in a decrease in Π at that point, followed by an increasing trend again afterward.

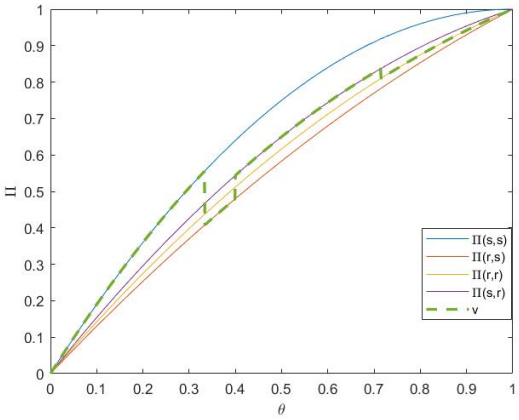


Figure 3.4: example 1

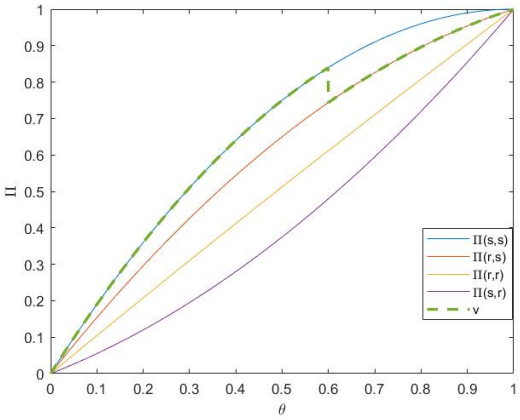


Figure 3.5: example 2

The two examples of the plots of Π over θ proves the statement:

(i) In a low-quality pool, a marginal increase in contestants' average quality increases selection efficiency.
(ii) In a medium-quality pool, a marginal increase in contestants' average quality may decrease selection efficiency.
(iii) In a high-quality pool, a marginal increase in contestants' average quality increases selection efficiency.

3.4. Increased number of contestants

We now investigate the impact of increasing the number of contestants, denoted as n, on the selection efficiency Π. To analyze this effect, we consider the case where $n = 3$ and find out the new Nash equilibrium, and then we compute the selection efficiency accordingly. In the following part, we will compute the utility functions for each type of agent under the four pure strategies (s,s), (s,r), (r,s), and (r,r), taking into account the increased number of contestants.

Let's proceed with the computation of the utility functions for the case where $n = 3$. We first examine the utility function $U_H(s,s)$, where both low types and high types adopt the safe strategy. In this case, there are several scenarios in which high types may win: 1) When three high types compete against each other (with probability $\theta)^2$), the agent has a 1/3 probability of winning. 2) When there is one low-type agent and two high-types competing (with probability $2 * \theta(1-\theta)$), the probability of high-types winning is 1/2. 3) When there are two low agents and only one high-type agent competing, the win probability is 1. Therefore we have:

$$U_H(s,s) = \theta^2 * \frac{1}{3} + 2 * \theta(1-\theta)\frac{1}{2} + (1-\theta)^2$$

Considering the utility function $U'_H(s,s)$, this is the case that the specific high-type agent chooses the risky strategy. Regardless of whom they compete with, the only way for this high-type agent to win is by obtaining the score z_4. Therefore we have:

$$U'_H(s,s) = y$$

Continuing with the same analytical approach, we proceed to calculate the win probabilities for player 1 of different types under various scenarios:

$$U_L(s,s) = \frac{1}{3}(1-\theta)^2$$

$$U'_L(s,s) = x$$

$$U_L(r,s) = \theta^2 x + 2\theta(1-\theta)x(1-x+\frac{x}{2}) + \frac{1}{3}(1-\theta)^2$$

$$U'_L(r,s) = (1-\theta)^2(1-x)^2$$

$$U_H(s,r) = \theta^2 \cdot \frac{1}{3} + 2 * \theta(1-\theta)\frac{1}{2}(1-x) + (1-\theta)^2(1-x)^2 = (\theta-1)^2 \cdot (x-1)^2 + \theta^2/3 + \theta * (\theta-1) * (x-1)^2$$

$$U'_H(s,r) = \theta^2 y + 2\theta(1-\theta)y[1-x+\frac{1}{2}x] + (1-\theta)^2[\frac{1}{2}(1-x)(1-y) + \frac{1}{2}xy + y(1-x)]^2$$

$$= ((\theta-1)^2(y-x+1)^2)/4 + \theta^2 y + 2\theta y(x/2-1)(\theta-1)$$

Recall from the previous section that when $n = 2$ and $0 < \theta < \min(1-2x, 2-2y)$, the strategy (s,s) constitutes a unique Bayesian Nash equilibrium. Under the same conditions where $0 < x < \frac{1}{2}$, $x < y < 1$, and $\theta < \min(1-2x, 2-2y)$, we observe that when the number of contestants increases to $n = 3$, the strategy (s,s) is no longer an equilibrium. This is because if $1/3 < x < 1/2$, $U'_L(s,s) > U_L(s,s)$. Consequently, if there are more contestants, then the low-type agent will become more risky.

Given this context, let's examine if the strategy (r,s) is the new equilibrium in the scenario where $n = 3$. Recall that for two players this is an equilibrium if $\theta < min(1-2x, 2-2y)$. If $x > 1/3$, therefore $\theta < \frac{1}{3}$, we have

$$U_L(r,s) - U'_L(r,s) > [1/3 - (1-x)^2 + 2\theta x](1-\theta)^2 + \theta^2 x > 0$$

and

$$U_H(s,r) - U'_H(s,r) = (\theta-1)^2(x-1)^2 - \theta^2 y - ((\theta-1)^2(y-x+1)^2)/4 + \theta^2/3 + \theta(\theta-1) * (x-1)^2 - 2\theta y(x/2-1)(\theta-1) > 0$$

This proves that (r,s) is the new Nash equilibrium.

This change in equilibrium strategy can subsequently influence the selection efficiency Π. After calculating the selection efficiencies for the strategies (s,s) and (r,s), denoted as $\Pi(s,s)$ and $\Pi(r,s)$ respectively, we observe the following:

$$\Pi(s,s) = \theta^3 + 2\theta^2(1-\theta) + 2\theta(1-\theta)^2$$

$$\Pi(r,s) = 2 \cdot \theta(\theta-1)^2 \cdot (x-1)^2 + \theta^3 + 2\theta^2(\theta-1)(x-1)^2$$

It becomes apparent that $\Pi(s,s) > \Pi(r,s)$ since $(x-1)^2 < 1$, which implies that in this case, the selection efficiency decreased when the number of agents increased.

This paper establishes several implicit conditions and focuses on the analysis of a contest involving two agents ($n = 2$). However, it does not explicitly address scenarios involving multiple agents ($n > 2$), which presents a limitation for further study. The implications and findings presented in the paper may not directly apply to contests with more than two agents, as the dynamics and strategic interactions can become more complex in such cases. The behavior and outcomes observed in two-agent contests may not necessarily generalize to contests involving more participants.

3.5. Summary

In this chapter, we reviewed the paper of Hvide and Kristiansen about risk and selection in a contest game. The mathematical analysis of Nash equilibrium and selection efficiency revealed surprising facts: in the job market, an increase in the number of contestants or an improvement in market quality may decrease selection efficiency. These insights from our mathematical analysis lead to the simulations in the next chapter.

4

Simulations of Selection Contest

In Chapter 3, we undertake a comprehensive mathematical analysis of a selection contest, with a particular emphasis on two key aspects: optimal risk-taking choices and selection efficiency. We identify pure Nash equilibrium under various parameter values. Drawing inspiration from the hawk-dove game example, we introduce different learning rules in simulations to explore whether players' strategies converge to pure Nash equilibrium through learning by playing. According to Robinson's theorem on fictitious play, this will happen if the players have infinite memory. But what if they have only limited memory? What if we have more than 2 contestants ($n > 2$)? The chapter delves into the simulation results derived from experiments conducted under four distinct scenarios: Agents without memory, Agents with limited memory, Multi-agent scenarios, and Multi-round contests. Additionally, we also examine the selection efficiency over θ under each scenario. Our objective is to conduct an analysis of these results and compare them with our earlier theoretical analysis, thereby providing a nuanced understanding of the dynamics at play in diverse contest scenarios.

4.1. Agents with no memory

First, we will present figures illustrating the variation in the proportion of risky agents, providing an analysis of example 1 (where $x = 1/3$, $y = 4/5$) and example 2 (where $x = 1/5$, $y = 1/4$) as outlined in Chapter 2. We will operate under the assumption that there are 100 agents within the market. As θ represents the quality of the market (the proportion of high-type agents), We check θ values from 0 to 1 in 10 steps (0.1, 0.2, 0.3, ..., 1), but the figures are plotted specifically for $\theta = 0.2, 0.5, 0.8$. These values represent the low-quality, medium-quality, and high-quality pool scenarios. In each round of the game, we begin by randomly pairing up the 100 agents, resulting in 50 pairs. A game is played within each pair, and the player who achieves the highest output emerges as the winner. The rules governing these game outputs align with the model defined in Chapter 3.

The dynamics of strategy adaptation in this game are as follows: the winning player retains their current strategy for the subsequent round, while the losing player is prompted to switch their strategy. In cases where a round ends in a tie, with both players achieving the same output, both individuals maintain their existing strategies for the next round. This iterative process continues, and we conduct this game 1000 times for each selected value of θ yielding a comprehensive set of results for our analysis. Over 1000 iterations, we compute the proportion of risky agents in both types respectively under low-quality, medium-quality, and high-quality markets. To mitigate fluctuations in the figures, a time window is employed, set at 50 within the context of 1000 time points.

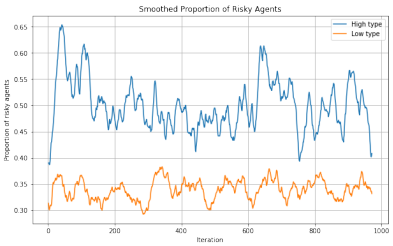


Figure 4.1: example 1(theta=0.2)

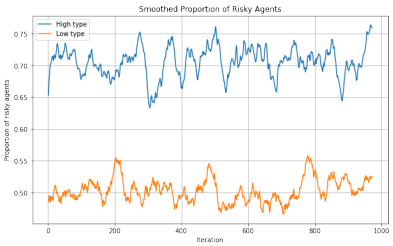


Figure 4.2: example 1(theta=0.5)

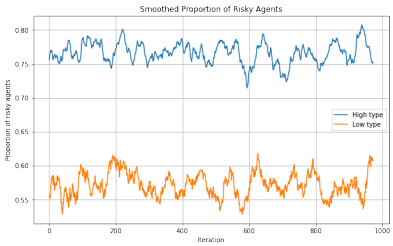


Figure 4.3: example 1(theta=0.8)

Figures 4.1 through 4.3 illustrate the pure Nash Equilibrium within the context of Example 1. As discussed in Chapter 3, the equilibrium strategy transitions from (s,s) to (s,r) and finally to (r,r) according to the theoretical analysis. Indeed, these figures show that variations in the parameter θ wield a substantial impact on the equilibrium strategy. Particularly, as depicted in Figures 4.1 to 4.3, a steady increase in the proportion of risk-taking agents becomes apparent as the parameter θ increases from 0.2 to 0.5 to 0.8, all while holding the total number of agents constant. For the high types, the proportion of risky ones rises from 0.5 to 0.75, and for the low types, it increases from 0.3 to 0.55. This indicates that both types of agents exhibit a propensity to embrace risky strategies swiftly. However, it is noteworthy that the percentage of risky agents fluctuates, even though we use the time window to smooth the figures, the proportion of risky agents never reaches 0 or 1. This makes the pure Nash Equilibrium never reached. Therefore, the simulation result is different from Fig 3.4 in the last Chapter which shows that equilibrium strategies change from (s,s) to (s,r) to (r,r) on different values of the parameter θ.

At the same time, it's also interesting to note that the results exhibit a heightened level of instability at the boundary points. For instance, when approximately 99% of the agents in the system are high-type, the behavior of low-type agents tends to exhibit a heightened level of unpredictability. To make the figure more readable, we use a time window (set at 2), and the reason for low types getting 0.5 is that he flips from 0 to 1 during the time window. This phenomenon can be attributed to the fact that, since all types of agents have a chance to win, they modify their strategies every time they encounter losses, introducing an element of unpredictability. When most agents are of a specific type, this phenomenon is particularly pronounced, leading to more frequent changes in the strategies of agents of the opposing type.

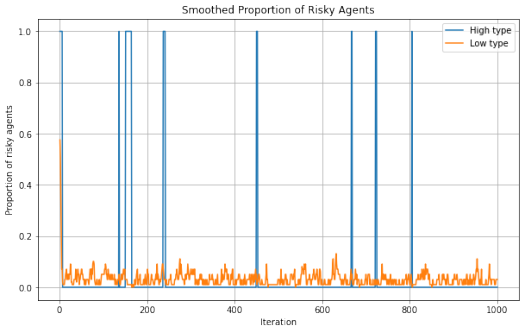


Figure 4.4: example 1(theta=0.01)

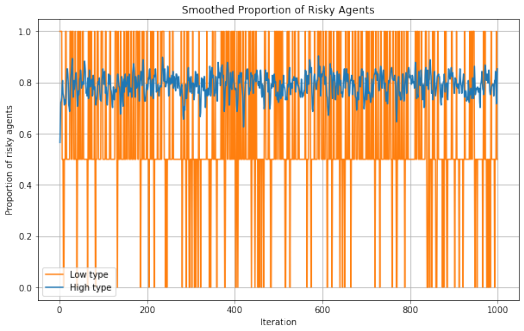


Figure 4.5: example 1(theta=0.99)(time window=2)

To illustrate the dynamics of the changes in the strategy, we reconfigure the plots with the x-axis representing the outcomes from the t iteration, and the y-axis representing the outcomes from the $t+1$ iteration. Despite potential randomness affecting the figures, excluding extreme edge points reveals a consistent pattern. The dense cluster of data points converges within a specific range, consistently indicating that an increase in θ leads to a rise in the proportion of risk-taking agents for both types.

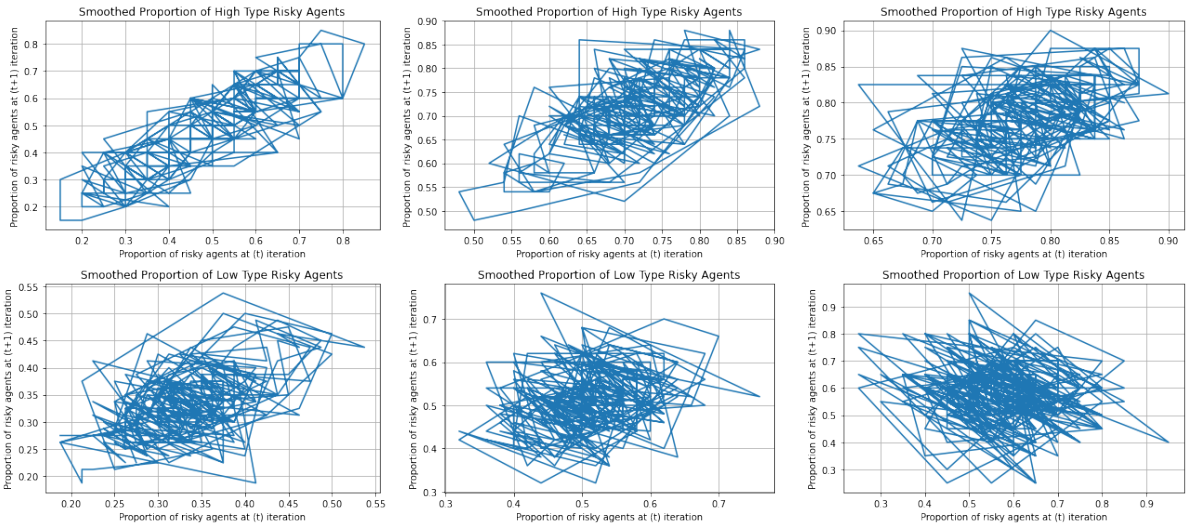


Figure 4.6: example 1(theta=0.2)

Figure 4.7: example 1(theta=0.5)

Figure 4.8: example 1(theta=0.8)

Conversely, when we examine example 2, as shown in Figures 4.9-4.11, the equilibrium strategies also depend on the values of the parameters (x, y) with the same θ. In this scenario, however, it is the low-type agents who demonstrate a greater inclination to transition to risky strategies at an earlier stage compared to their high-type contestants, and the equilibrium moves from (s, s) to (r, s), in accordance with the findings presented in Figure 3.3.

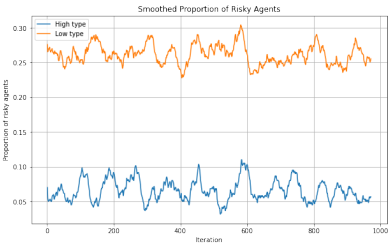


Figure 4.9: example 2(theta=0.2)

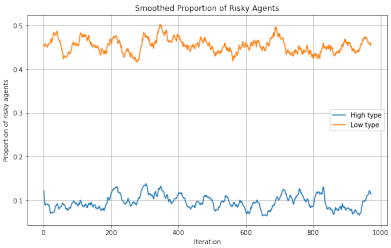


Figure 4.10: example 2(theta=0.5)

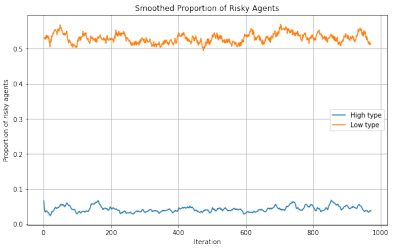


Figure 4.11: example 2(theta=0.8)

Another aspect of our analysis is how market quality (θ) influences the win probability of the high-type agent (selection efficiency), which is computed by the number of winning high types/ number of high types. As illustrated in figures 4.12 to 4.14, a consistent trend emerges. The selection efficiency increases as θ rises from 0.2 to 0.8, moving from 0.43 to 0.7 and then to 0.88. Similarly, from Figures 4.15 to 4.17, the trend persists from 0.5 to 0.65 and then to 0.85. This trend persists across low-quality, medium-quality, and high-quality contestant pools. Surprisingly, contrary to the theoretical analysis in Chapter 3, which suggested a potential decrease in selection efficiency in a medium-quality pool with a marginal increase in contestants' average quality, the figures consistently show an overall increase in selection efficiency with elevated contestant quality (θ), leading to an increasing win probability of the high-type agent. This difference may arise due to the influence of randomness, leading to strategies far from pure Nash Equilibrium and instability in the choices made by both types. In this scenario, low types cannot learn to use the risky strategy, and the random strategy selection reduces their chances of manipulating the market.

These graphs offer valuable insights into how changes in contestant quality (θ) impact the equilibrium strategy and, more importantly, impact the selection efficiency, shedding light on the distinct dynamics at play in both example 1 and example 2. The observed randomness and absence of convergence can be attributed to the rules of the game, where an agent adjusts their strategy in every iteration following a loss in the game. Therefore, we proceeded to conduct the subsequent experiment, examining the implications of allowing agents to learn from their prior actions, thereby obviating the necessity for them to adjust their strategy following each instance of losing in the game.

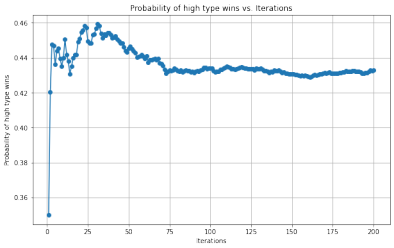


Figure 4.12: example 1(theta=0.2)

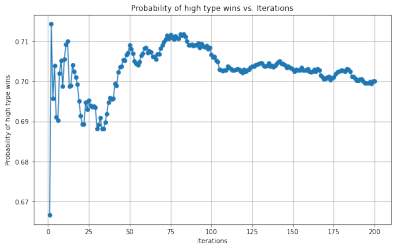


Figure 4.13: example 1(theta=0.5)

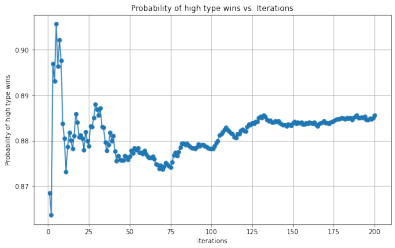


Figure 4.14: example 1(theta=0.8)

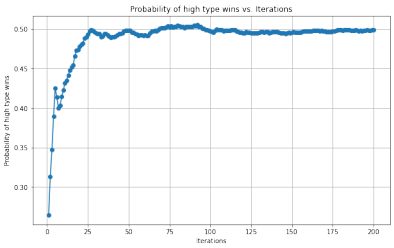


Figure 4.15: example 2(theta=0.2)

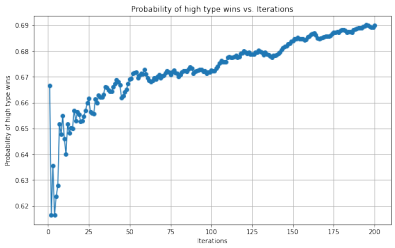


Figure 4.16: example 2(theta=0.5)

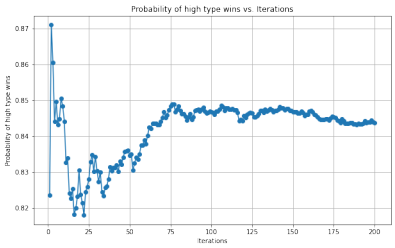


Figure 4.17: example 2(theta=0.8)

4.2. Agent with limited memory

A modification of our assumptions regarding agent behavior occurs. In the last experiment, the large fluctuations make the Nash equilibrium impossible to reach, because agents flip between risky and safe moves frequently. In this experiment, we will enable agents to learn not only from their most recent interaction but also from the preceding steps in the game. Here are the results of the experiments involving three-step memory for both examples 1 and 2. The experiments were conducted with varying values of θ from 0 to 1. In these experiments, the parameters x and y are also set to $1/3$ and $4/5$, and for example 2, they were set to $1/5$ and $1/4$. Agents in this scenario possess a memory of the last three results, and if they both lose, they will modify their strategy based on this limited memory. Figures 4.18 to 4.23 vividly illustrate the transformation of the proportion of risky agents as a function of θ, the proportion of high-type agents within the market. These figures offer valuable insights into the relationship between θ and the strategic behaviors of agents.

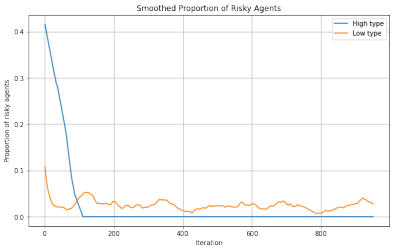


Figure 4.18: example 1(theta=0.2)

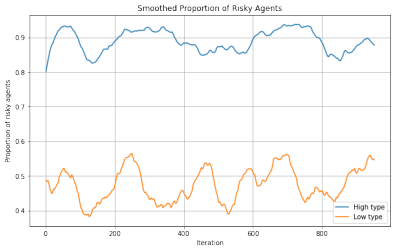


Figure 4.19: example 1(theta=0.5)

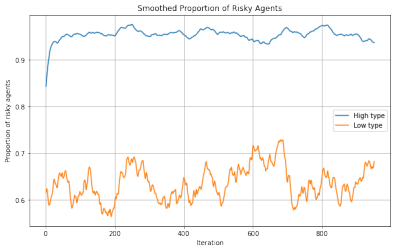


Figure 4.20: example 1(theta=0.8)

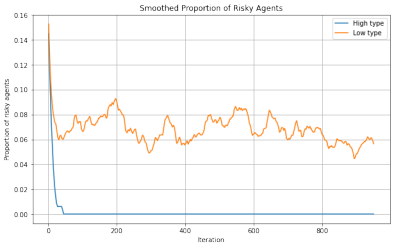


Figure 4.21: example 2(theta=0.2)

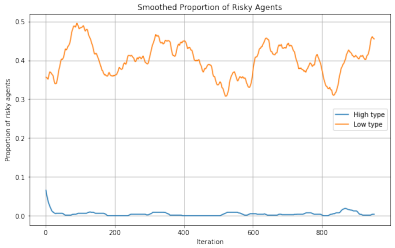


Figure 4.22: example 2(theta=0.5)

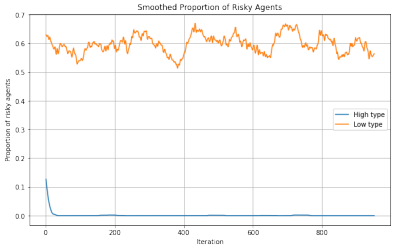


Figure 4.23: example 2(theta=0.8)

When agents possess memory, the figures reveal remarkable consistency in the results. The agents still do

not reach pure NE, in line with our assumption, which differs from George Brown's notion in Fictitious Play where infinite learning steps lead to convergence to the game's value. However, comparing these figures to example 1 in Chapter 3, the result gets closer to pure Nash Equilibrium. In example 1, within a low-quality pool, there are almost no risky agents, and the strategy equilibrium (s, s) stands as a Nash equilibrium. Transitioning to a medium-quality pool, approximately 90% of high types opt for the risky strategy, while only 50% of low types choose the safe strategy. In a high-quality pool, about 95% of high types select the risky strategy, and around 60% of low types opt for the safe strategy. In example 2, the high types keep the safe strategy, while an increasing number of low types choose the risky strategy.

Furthermore, the observed outcomes indicate a potential decrease in selection efficiency as the market quality improves. This is because all agents are converging to their optimal strategies. When the Nash Equilibrium changes in the medium-quality market, a decline in selection efficiency is shown. As an illustrative instance, in Example 1, the selection efficiency declines from 0.65 to 0.6 as θ transitions from 0.2 to 0.3 and then increases up to 0.88 as $\theta = 0.8$. This correspondence with our mathematical analysis in Chapter 3 that in the medium-quality pool, a marginal increase in θ may decrease selection efficiency. In the case of example 1, the figures are as follows:

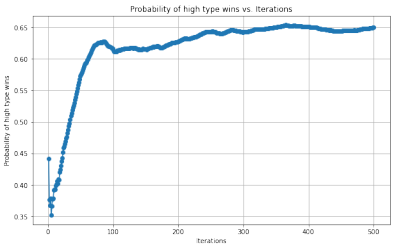


Figure 4.24: example 1(theta=0.2)

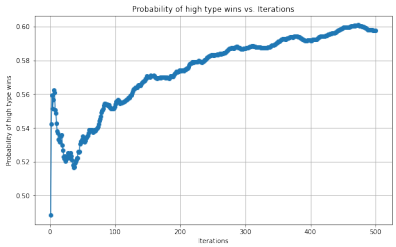


Figure 4.25: example 1(theta=0.3)

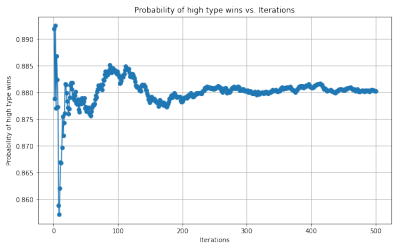


Figure 4.26: example 1(theta=0.8)

In example 2, When θ changes from 0.3 to 0.4 and 0.8, selection efficiency initially decreases from 0.75 to 0.7 and then undergoes an increase to 0.83.

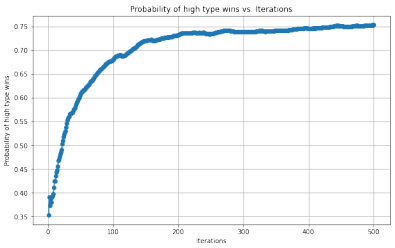


Figure 4.27: example 2(theta=0.3)

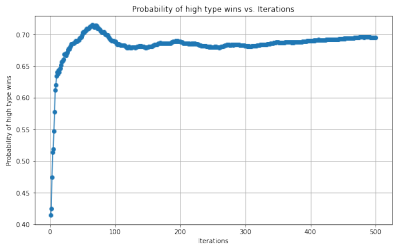


Figure 4.28: example 2(theta=0.4)

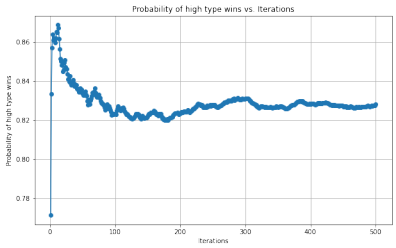


Figure 4.29: example 2(theta=0.8)

The figures for agents with memory show that: 1) The players do not converge to pure Nash Equilibrium, but get closer to pure Nash Equilibrium than when they have no memory because they get closer to their optimal strategy. 2) There are situations where, as θ increases, the corresponding selection efficiency (Π) may decrease.

4.3. Multi-agent

Several factors have the potential to influence the outcomes of our experiments. One of these factors is the scenario where more than two agents compete. As a consequence, we will proceed to undertake a new series of simulations. We examine the scenarios involving more than two agents within a single competition. We conducted experiments as follows: simulate a market consisting of 100 agents, and divide them into 25 groups, each containing 4 agents. In other words, this is the case for $n = 4$, which was not considered by Hvide and Kristiansen, 2003. The group's only winner(s) maintain their strategy while the others modify theirs. This experiment was conducted for three different values of θ: 0.2, 0.5, and 0.8.

It is noteworthy that agents never reach a Nash equilibrium under the current conditions. Additionally, it's pointed out that there is a slight trend regarding risk-taking behavior among the agents. This implies that the

strategies adopted by the agents do not consistently shift towards risk-taking as θ increases. In comparison with the scenario where $n = 2$, the results become intriguing due to the observed decrease in the proportion of risk-taking agents as the value of θ increases, as depicted in the accompanying figures. This shift in the equilibrium introduces a unique dynamic to our analysis. This might be because the Nash Equilibrium is no longer pure if $n = 4$. Since there is only 1 winner and 3 losers, more players change their behavior and the results are more volatile.

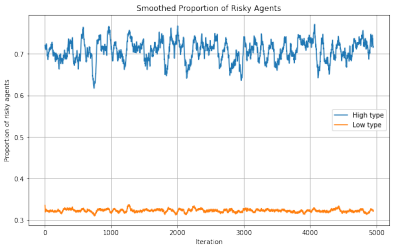


Figure 4.30: example 1(theta=0.2)

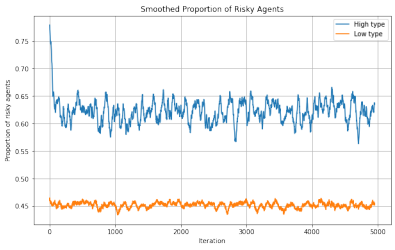


Figure 4.31: example 1(theta=0.5)

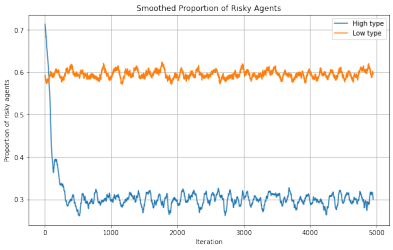


Figure 4.32: example 1(theta=0.8)

In terms of selection efficiency, a noteworthy observation is a decrease when compared to the scenario where $n = 2$. For instance, in example 1, under the same conditions, when $\theta = 0.2$ and $n = 4$, Π is approximately 0.25, whereas for $n = 2$, Π is 0.43. This trend is consistent for θ values at 0.5 and 0.8 as illustrated in figures 4.33-4.35. This demonstrates that increasing the number of contestants, starting from a smaller value may negatively impact selection efficiency. This might be because of the negative equilibrium effect of increases in n, more agents change their behavior in one round, as long as they don't get the highest score. This trend towards the risky strategy provides low types with more opportunities to attain high scores and manipulate the market. This might cause a decrease in selection efficiency.

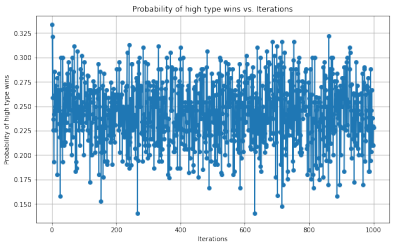


Figure 4.33: example 1(theta=0.2)

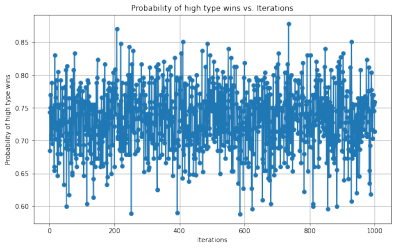


Figure 4.34: example 1(theta=0.5)

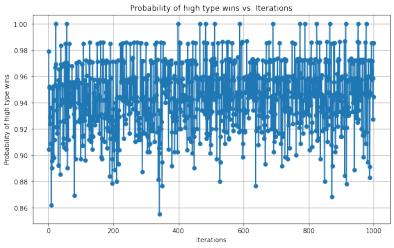


Figure 4.35: example 1(theta=0.8)

4.4. Multi-round selection

In this section, we will compare two mechanisms within a multi-round selection game, evaluating their impact on selection efficiency.

The first mechanism involves the selection of 50 pairs from a pool of 100 agents, engaging in pairwise competitions. Unlike the previous mechanism, the defeated agent doesn't have the chance to alter their strategy but is eliminated from the game directly. When there's a draw, the two contestants involved will proceed to the next round. The victorious agent advances to subsequent rounds, competing with other winners, and this two-round structure repeats. If in the second round, the total number of agents is odd, we will Select one agent directly for the next round.

The second mechanism entails the creation of 25 groups, each composed of 4 agents, competing within their respective groups for a single round. Following the competition, 25 agents are expected to remain in each mechanism, allowing for a comparative analysis of selection efficiency.

The result is as follows:

θ	Mechanism 1	Mechanism 2
0.1	0.09840	0.11039
0.2	0.19519	0.22800
0.5	0.65356	0.57742
0.8	0.92657	0.92518

Table 4.1: Selection Efficiency of Two Mechanisms

The results show that the Multi-round mechanism doesn't always hold an advantage. In the low-quality pool, mechanism 2 exhibits better results, while in the middle and high-quality pool, mechanism 1 has the advantage for selection efficiency.

4.5. Summary

In this chapter, we mainly undertake four simulation experiments applying various learning methods to validate the findings of our mathematical analysis. Although a Nash Equilibrium is not strictly achieved, the results demonstrate a convergence towards Nash Equilibrium when agents have limited memory. The results also show the negative equilibrium effect of increases in number of contestants (n) or market quality (θ) on selection efficiency (Π). When $n = 2$, the efficiency loss does not occur as θ increases when contestants have no memory. However, some efficiency loss is observed when contestants have limited memory. Moreover, an increase in n can lead to a decrease in efficiency. Notably, we find that in a low-quality pool, single-round selection has an advantage, while in a higher-quality market, multi-round selection proves to have an advantage.

5

Conclusion

In this study, we explored a competition game within game theory, where the key variable is not the effort but rather risk-taking. Unlike conventional competitions, high quality in this context is associated with a propensity for higher-risk strategies, leading to potential high scores. Surprisingly, mathematical analyses revealed counterintuitive outcomes. As market quality improves or the total number of agents increases, the winning rates of high-type agents may decrease. This insight extends to labor markets, providing valuable perspectives for both candidates and recruiters.

Applying this competition paradigm to labor markets has implications for job seekers and employers. Candidates benefit from understanding the balance between risk and reward in decision-making, while recruiters need to consider the efficiency of selection mechanisms, which may not follow a straightforward pattern with market size or quality. Our mathematical analysis in Chapter 3 shows that selection efficiency might be non-monotone in market size or market quality.

The simulation experiments provided further depth to the investigation, introducing the concept of learning rules and replicator dynamics to improve the alignment of outcomes with Nash equilibrium. This proposed mechanism, derived from agents gaining experience from their behavior, has been demonstrated to align outcomes more closely with Nash equilibrium, thereby enhancing the optimal result. At the same time, in this scenario, a decrease in selection efficiency is observed. We also explore a scenario with $n = 4$ and observe a potential decrease in efficiency compared to $n = 2$.

Additionally, the exploration of multi-round screening mechanisms offers practical insights into recruitment strategies. The unexpected superiority of single-round screening challenges assumptions and emphasizes the need to tailor selection processes to specific market characteristics.

In all, this study contributes a novel perspective to the competition game. The findings encourage a reevaluation of traditional assumptions and offer practical implications for decision-makers in competitive environments, particularly within labor markets.

A

Matlab Code

A.1. Chapter 3-Equilibrium Strategies

```
% Define ranges
x_range = 0:0.05:1;
y_range = 0:0.05:1;
z_range = 0:0.05:1;

[x, y, z] = meshgrid(x_range, y_range, z_range);

% Define inequalities
region1 = y < 1 - z/2 & x < 0.5 .* (1 - z) & y > x;
region2 = y > 1 - z/2 & x < (1 - z .* y)./ (2 - z) & y > x;
region3 = y < (1 - (1 - z) .* x) ./ (1 + z) & x > (1 - z) ./ 2 & y > x;
region4 = y > (1 - (2 - z) .* x) ./ z & y > x;

% Create a scatter plot for data points within each region
figure;
scatter3(x(region1), y(region1), z(region1), 'r.');
hold on;
scatter3(x(region2), y(region2), z(region2), 'y.');
scatter3(x(region3), y(region3), z(region3), 'b.');
scatter3(x(region4), y(region4), z(region4), 'g.');

% Set labels and title
xlabel('x');
ylabel('y');
zlabel('\theta');
title('Equilibrium Strategie');

% Create shaded regions for each condition
alpha(0.3);

% Add color bar
legend('Region (s, s)', 'Region (s, r)', 'Region (r, s)', 'Region (r, r
    )');

% View in 3D
view(3);
grid on;
hold off;
```

Here's the code for risk-taking strategies over θ of example 1:

```
syms x y theta

x = 1/3;
y = 4/5;

theta = 0:0.001:1;

f = 2 * theta - theta.^2;
z = 2 * (1/2 * theta.^2 + (1 - theta) .* theta .* (1 - x));
g = 2 * (1/2 * theta.^2 + (1 - theta) .* theta * y);
w = 2 * (1/2 * theta.^2 + theta .* (1 - theta) .* (1/2 * x * y + y * (1
    - x) + 1/2 * (1 - x) * (1 - y)));

v = zeros(size(theta));

for i = 1:length(theta)
    if theta(i) < 1 - 2 * x
        v(i) = f(i);
    elseif (theta(i) >= 1 - 2 * x) && (theta(i) < 2 - 2 * y)
        v(i) = z(i);
    elseif (theta(i) >= 2 - 2 * y) && (theta(i) < (1 - 2 * x) / (y - x)
       )
        v(i) = g(i);
    elseif theta(i) >= (1 - 2 * x) / (y - x)
        v(i) = w(i);
    end
end

figure;
plot(theta, f, theta, z, theta, w, theta, g);
hold on;
plot(theta, v, '--', 'LineWidth', 2); % <-- Use '--' for dashed line
   and 'LineWidth', 2 for thickness
hold off;
xlabel('\theta');
ylabel('\Pi');
legend('\Pi(s,s)', '\Pi(r,s)', '\Pi(r,r)', '\Pi(s,r)', 'v');
```

B

Python Code

B.1. Chapter 2-Hawk-Dove Game

Firstly, we define the hawk dove game.

```
import random
class Bird:
    def __init__(self, strategy):
        self.strategy = strategy
        self.fitness = 10

    def contest(self, opponent, v, c):

        # both hawks --> 50:50 battle

        if self.strategy == opponent.strategy == "hawk":
            if random.randint(0, 1) == 1:
                self.fitness = self.fitness + v
                opponent.fitness = opponent.fitness - c
            else:
                self.fitness = self.fitness - c
                opponent.fitness = opponent.fitness + v

        # hawk meets dove

        elif self.strategy == "hawk" != opponent.strategy:
            self.fitness = self.fitness + v
            opponent.fitness = opponent.fitness
        elif self.strategy == "dove" != opponent.strategy:
            self.fitness = self.fitness
            opponent.fitness = opponent.fitness + v

        # both doves --> share the resource

        else:
            self.fitness = self.fitness + v/2
            opponent.fitness = opponent.fitness + v/2

    def spawn(self):
        """
        Allow a small chance of mutation to flip the strategy
        Otherwise, return offspring of the same type
```

```
        """

        mutation = random.randint(0, 1000) > 999
        if mutation:
            if self.strategy == "dove":
                return Bird("hawk")
            else:
                return Bird("dove")
        else:
            return Bird(self.strategy)
```

Then we pair up the birds, make them compete, and produce next-generation weighted by fitness

```
from bird import Bird
import random
import numpy as np
import pandas as pd
import matplotlib

def initialise():
    #Create a population of birds - all dove to begin

    birds = []
    for _ in range(999):
        birds.append(Bird("dove"))

    # Adding a single Hawk to the population
    birds.append(Bird('hawk'))

    #for _ in range(100):
     #   birds.append(Bird("dove")*99,Bird('hawk'))

    return (birds)


def timestep(birds, value, cost):
    #Pair up the birds, make them compete
    #Then produce next generation, weighted by fitness

    next_generation = []

    random.shuffle(birds)

    for _ in range(1000):

        # pair up random birds to contest
        a, b = random.sample(birds, 2)
        a.contest(b, value, cost)

    # generate next generation
    fitnesses = [bird.fitness for bird in birds]

    draw = random.choices(birds, k=1000, weights=fitnesses)
    next_generation = [bird.spawn() for bird in draw]

    return next_generation
```

```
def main():

    birds = initialise()

    rows = []

    V = 5 ; C = 6

    for _ in range(1000):

        # add the counts to a new row
        strategy = [bird.strategy for bird in birds]
        n_hawks = strategy.count("hawk")
        n_doves =  strategy.count("dove")
        row = {'n_hawks': n_hawks, 'n_doves': n_doves}
        rows.append(row)

        # run the timestep function
        birds = timestep(birds, V, C)

    df = pd.DataFrame(rows)
    df.to_csv('simulation.csv')
    fig = df.plot(y=["n_hawks", "n_doves"]).get_figure()
    fig.savefig('simulation.pdf')

if __name__ == "__main__":
    main()
```

B.2. Chapter 2-Hawk-Dove Game(mutation)

```
import random
import matplotlib.pyplot as plt

def hawk_dove_game(population):
    # Shuffle the population to form pairs
    random.shuffle(population)

    payoffs = []

    # Iterate through pairs and calculate payoffs
    for i in range(0, len(population), 2):
        strategy_1 = population[i]
        strategy_2 = population[i + 1]

        payoff_1, payoff_2 = 0, 0

        if strategy_1 == "Hawk" and strategy_2 == "Hawk":
            #payoff_1, payoff_2 = -10, -10
            payoff_1, payoff_2 = -1, -1
        elif strategy_1 == "Hawk" and strategy_2 == "Dove":
            #payoff_1, payoff_2 = 100, 0
            payoff_1, payoff_2 = 10,0
        elif strategy_1 == "Dove" and strategy_2 == "Hawk":
            #payoff_1, payoff_2 = 0, 100
            payoff_1, payoff_2 = 0,10
        elif strategy_1 == "Dove" and strategy_2 == "Dove":
```

```
            #payoff_1, payoff_2 = 50, 50
            payoff_1, payoff_2 = 5,5

        payoffs.append((payoff_1, payoff_2))

    # Update strategies based on payoffs
    for i in range(0, len(population), 2):
        payoff_1, payoff_2 = payoffs[i // 2]

        if payoff_1 > payoff_2:
            # Change the type of the loser (Dove) to Hawk
            population[i + 1] = population[i]
        elif payoff_2 > payoff_1:
            # Change the type of the loser (Dove) to Hawk
            population[i] = population[i + 1]

    return population

def calculate_proportion(population):
    # Calculate the proportion of Hawks in the population
    num_hawks = population.count("Hawk")
    total_birds = len(population)
    proportion_hawks = num_hawks / total_birds

    return proportion_hawks

def print_population_with_indices(population):
    for i, strategy in enumerate(population):
        print(f"Element {i + 1}: {strategy}")

def plot_population_proportion(iterations, initial_population):
    proportions = []

    for iteration in range(iterations):
        print(f"Iteration {iteration + 1}:")
        print_population_with_indices(initial_population)

        initial_population = hawk_dove_game(initial_population)

        # Calculate the proportion of Hawks in the population
        proportion_hawks = calculate_proportion(initial_population)

        proportions.append(proportion_hawks)

    # Plotting
    plt.plot(range(1, iterations + 1), proportions, marker='o')
    plt.xlabel('Iterations')
    plt.ylabel('Proportion of Hawks')
    plt.title('Change in Proportion of Hawks Over Iterations')
    plt.show()

# Initial population: 99 Doves and 1 Hawk
initial_population = ["Dove"] *9+ ["Hawk"]

# Number of iterations
num_iterations = 50  # You can adjust this number as needed
```

```
plot_population_proportion(num_iterations, initial_population)
```

B.3. Chapter 4-Agents with no memory

```
import random
import matplotlib.pyplot as plt
import numpy as np

# Define 'scores' function to represent the outputs.
n=100
theta=0.2
iteration=1000
high=theta*n
def update_scores(population_subset):
    updated_scores = []
    for i, value in enumerate(population_subset):
        if i < high:
            # For the first theta elements (high type agent)
            if value == 0: #safe strategy
                updated_scores.append(3) #output z3
            else: #risky strategy
                if random.random() < 0.8: #y=1/4 #y=0.8
                    updated_scores.append(4) #output z4
                else:
                    updated_scores.append(1) #output z1
        else:
            # For the last 50 elements (low-type agent)
            if value == 0: #safe strategy
                updated_scores.append(2) #output z2
            else: #risky strategy
                if random.random() < 1/3: #x=1/5 #x=1/3
                    updated_scores.append(4) #output z4
                else:
                    updated_scores.append(1) #output z1
    return updated_scores

# Set up array 'population' to represent strategy, 0 means safe and 1 means risky
#Initialize 'population' array with 100 elements (0 or 1) randomly.
population = [random.randint(0, 1) for _ in range(100)] #100 agents
scores = update_scores(population)
print (population)
score_indices = list(range(1, 101))  # Indices of scores (1 to 100).
high=theta*n
low=(1-theta)*n
elements=int(theta*n)
sum_population_high = []  # Sum of the first theta elements
sum_population_low = []   # Sum of the last 1-theta elements

# Perform updates 100 times.
for update_step in range(iteration):
    # Select 50 pairs randomly and compare the outputs (scores)
    random.shuffle(score_indices)
    pairs_indices = [(score_indices[i], score_indices[i + 1])\
    for i in range(0, 100, 2)]
    pair_comparisons = []
```

```
high_type_agent=population[:elements]
low_type_agent=population[-(n - elements):]

# Calculate the sum of the first theta  elements in 'population'
# Calculate the sum of the last 1-theta elements in 'population'
sum_high_agents = high_type_agent.count(1)
sum_low_agents = low_type_agent.count(1)
sum_population_high.append(sum_high_agents/high)
sum_population_low.append(sum_low_agents/low)



for compared_indices in pairs_indices:
    score1 = scores[compared_indices[0] - 1]  # Subtract 1 for 0-based indexing
    score2 = scores[compared_indices[1] - 1]

    if score1 == score2:
        # Handle case where scores are equal
        pair_comparisons.append({
            "element1_index_in_scores": compared_indices[0],
            "element2_index_in_scores": compared_indices[1],
            "element1_value": score1,
            "element2_value": score2,
            "lower_index_in_scores": None,  # Set lower_index to None
        })
    else:
        lower_index = compared_indices[0] if score1 < score2\
        else compared_indices[1]
        higher_index = compared_indices[1] if score1 < score2\
        else compared_indices[0]

        pair_comparisons.append({
            "element1_index_in_scores": compared_indices[0],
            "element2_index_in_scores": compared_indices[1],
            "element1_value": score1,
            "element2_value": score2,
            "lower_index_in_scores": lower_index,
            "higher_index_in_scores": higher_index,
        })

# Update strategy ('population' array) based on the comparison results.
population_new = population.copy()
for comparison in pair_comparisons:
    lower_index = comparison.get("lower_index_in_scores")

    if lower_index is not None: # the scores are different
        corresponding_element = population[lower_index - 1]
        if corresponding_element == 0: #if the previous strategy is safe
            population_new[lower_index - 1] = 1  #change to risky
        else:
            population_new[lower_index - 1] = 0  # change to safe

# Collect the index of changed elements in 'population' starting from 1.
changed_indices = [i + 1 for i in range(100) if\
population[i] != population_new[i]]
```

```
    # Update 'population' and 'scores' for the next iteration.
    population = population_new
    scores = update_scores(population)
    sum_pop=sum(population_new)
    print(changed_indices)
    print(pair_comparisons)
    print(population_new)
    print(f"Iteration {update_step + 1}: {scores}")
    print(sum_pop)
    #print(population)

# Lists to store data for plotting.
proportion_high = []
proportion_low = []

# Time window for calculating the moving average
time_window = 50
# Apply moving average to the proportions.
smoothed_proportion_high = np.convolve(sum_population_high,np.ones(time_window)\
/time_window,mode='valid')
smoothed_proportion_low = np.convolve(sum_population_low,np.ones(time_window) \
/ time_window,mode='valid')

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(smoothed_proportion_high) + 1), smoothed_proportion_high,\
linestyle='-',markersize=2, label='High type')
plt.plot(range(1, len(smoothed_proportion_low) + 1), smoothed_proportion_low, \
linestyle='-',markersize=2, label='Low type')
plt.title("Smoothed Proportion of Risky Agents")
plt.xlabel("Iteration")
plt.ylabel("Proportion of risky agents")
plt.legend()
plt.grid(True)
plt.show()
```

And the code to compute selection efficiency is that:

```
plt.figure(figsize=(10, 6))
# Create x-axis values (iterations)
iterations = list(range(1, iteration + 1))
# Create y-axis values (higher_index_probabilities)
plt.plot(iterations, probability_higher_index, marker='o', linestyle='-')

# Add labels and title
plt.xlabel('Iterations')
plt.ylabel('Probability of high type wins')
plt.title('Probability of high type wins vs. Iterations')

# Show the plot
plt.grid(True)
plt.show()
```

B.4. Chapter 4-Agents with memory

```
import random
import matplotlib.pyplot as plt
import numpy as np
```

```
def update_scores(population_subset):
    updated_scores = []
    for i, value in enumerate(population_subset):
        if i < high: # For the first theta elements (high type agent)
            if value == 0: #safe strategy
                updated_scores.append(3) #output z3
            else: #risky strategy
                if random.random() < 0.8: #y=1/4 #y=0.8
                    updated_scores.append(4) #output z4
                else:
                    updated_scores.append(1) #output z1
        else:
            # For the last 50 elements (low-type agent)
            if value == 0: #safe strategy
                updated_scores.append(2) #output z2
            else: #risky strategy
                if random.random() < 1/3: #x=1/5 #x=1/3
                    updated_scores.append(4) #output z4
                else:
                    updated_scores.append(1) #output z1
    return updated_scores

n = 100
theta = 0.9
iterations = 1000
high = int(theta * n)
low = n - high
elements = int(theta * n)
probability_higher_index = []
z=4
# Initialize 'population' array with 100 agents.
population = [random.randint(0, 1) for _ in range(n)]
common_smaller_scores = []
# Initialize 'scores' array based on 'population'.
scores = update_scores(population)
score_indices = list(range(1, 101))
sum_population_high = []
sum_population_low = []
agent_highest_indice=[]
# Initialize lists to track the results of each score in every two iterations.
previous_score_results = []
current_score_results = []

# Initialize a list to track which elements are compared and need to be changed.
#elements_compared = [False] * n

# Initialize a list to store comparisons.
pair_comparisons = []
# Perform updates 'iterations' times.
for update_step in range(iterations):
    print(update_step)
    # Select 50 pairs randomly from 100 agents and compare the scores.
    random.shuffle(score_indices)
    #print(score_indices)
    pairs_indices = [(score_indices[i], score_indices[i + 1])\
```

```
for i in range(0, 100, 2)]
#print(pairs_indices)

# Iterate over pairs and compare the scores.
for compared_indices in pairs_indices:
    score1 = scores[compared_indices[0] - 1]
    score2 = scores[compared_indices[1] - 1]

    if score1 != score2:
        smaller_index = compared_indices[0] if score1 < score2\
        else compared_indices[1]
        higher_index = compared_indices[1] if score1 < score2\
        else compared_indices[0]

        pair_comparisons.append({
            "element1_index_in_scores": compared_indices[0],
            "element2_index_in_scores": compared_indices[1],
            "element1_score": score1,
            "element2_score": score2,
            "lower_index_in_scores":smaller_index,
            "higher_index_in_scores": higher_index,
        })
        agent_highest_indice.append(higher_index)

    else:
        pair_comparisons.append({
            "element1_index_in_scores":compared_indices[0],
            "element2_index_in_scores": compared_indices[1],
            "element1_value": score1,
            "element2_value": score2,
            "lower_index_in_scores": None,# Set lower_index to None
        })


        #print(f"Iteration {update_step + 1}: Agent {smaller_index}\
        (Score: {score1}) compared with Agent {higher_index}\
        (Score: {score2 if smaller_index == agent1 else score1})")
print(pair_comparisons)

highest_indices = [j for j in agent_highest_indice if j <= elements]
#highest_indices = [j["higher_index_in_scores"] for j in pair_comparisons\
if "higher_index_in_scores" in j and j["higher_index_in_scores"] <= elements]
probability = len(highest_indices) / len(agent_highest_indice)
probability_higher_index.append(probability)

#To create the memory
# Record the results of each score comparison.
population_new = population.copy()
for comparison in pair_comparisons:
    small_index = comparison["lower_index_in_scores"]
    #element2_index = comparison["element2_index_in_scores"]
    if small_index is not None:
        small_index = int(small_index)
    #element2_index = int(element2_index)
        current_score_results.append(small_index)
```

```
    # Check if this is the second iteration of the pair comparison.
        if update_step % z == 1:
            common_smaller_scores = []
        # Identify the scores that are the smaller ones in both iterations
            common_smaller_scores= [i for i in previous_score_results\
            if i in current_score_results]
            previous_score_results=[]
            current_score_results=[]
        # Update the population for agents with common smaller scores.
            for agent_index in common_smaller_scores:
                if agent_index is not None:
                    population_new[agent_index-1] =\
                    1 - population_new[agent_index-1]  # Flip the strategy

    # Reset the elements_compared list and pair_comparisons for the next iteration.
    #elements_compared = [False] * n
    pair_comparisons = []

    # Copy the current score results to the previous results for the next iteration.
    previous_score_results = current_score_results.copy()
    current_score_results=[]

    # Divide 'population' into high and low type agents based on the threshold.
    high_type_agent = population[:int(theta * n)]
    low_type_agent = population[int(theta * n):]

    # Calculate the sum of high and low type agents in 'population'.
    sum_high_agents = sum(high_type_agent)
    sum_low_agents = sum(low_type_agent)
    sum_population_high.append(sum_high_agents / (theta * n))
    sum_population_low.append(sum_low_agents / ((1 - theta) * n))
    population = population_new
    scores = update_scores(population)

    # Print the results for the current iteration.
    #print(f"Iteration {update_step+1}:")
    print(f"Population: {population}")
    print(common_smaller_scores)
    common_smaller_scores=[]
    print(f"Iteration {update_step + 1}: {scores}")

# Print the final population.
print("Final Population:", population)
```

B.5. Chapter 3-Multi-agent

```
n = 100
theta = 0.2
iterations = 1000
high = int(theta * n)
low = n - high
elements = int(theta * n)
probability_highest_index = []
# Initialize 'population' array with 100 agents.
population = [random.randint(0, 1) for _ in range(n)]

# Lists to store data for plotting.
```

```
sum_population_high = []
sum_population_low = []

shuffled_indices = list(range(n))
random.shuffle(shuffled_indices)

# Perform updates 'iterations' times.
for update_step in range(iterations):
    # Divide shuffled indices into 25 groups with 4 elements each.
    group_indices = [shuffled_indices[i:i+4] for i in range(0, n, 4)]
    #print(group_indices)

    # Update 'scores' based on the current 'population'.
    scores = update_scores(population)

    # Compare the scores within each group and find the lowest score.
    min_scores = []
    max_scores = []

    for group in group_indices:
        group_scores = [scores[i] for i in group]
        #print(group_scores)
        min_score = min(group_scores)
        min_scores.append(min_score)
        max_score=max(group_scores)
        max_scores.append(max_score)


    # Identify the indices of agents with the highest score.
    agent_indices = []
    highest_index=[]
    agent_highest_indice=[]
    highest_scores_list=[]
    highest_scores_count_list=[]
    All_highest_scores_count_list=[]

    for group_index, group in enumerate(group_indices):
        lowest_score = min_scores[group_index]
        print(lowest_score)
        highest_score=max_scores[group_index]
        print(highest_score)
        lowest_indices = [i for i, score in\
        enumerate(group_scores) if score == lowest_score]
        highest_indices = [i for i, score in\
        enumerate(group_scores) if score == highest_score]

        agent_indices.extend([group[i] for i in lowest_indices])
        agent_highest_indice.extend([group[i] for i in highest_indices])

        All_highest_scores=[score for i, score in enumerate(group_scores)\
        if score == max(group_scores)]

        highest_scores = [score for i, score in enumerate(group_scores)\
        if score == max(group_scores) and i <= elements]
        highest_scores_count = len(highest_scores)
        All_highest_scores_count = len(All_highest_scores)
```

```
    #print(len(agent_highest_indice))
    #print(agent_highest_indice)
    highest_indice_count = [j for j in agent_highest_indice if j <= high]
    probability_higher=len(highest_indice_count)/len(agent_highest_indice)
    #probability_higher_index.append(probability_higher)
    #print(agent_highest_indice)

    probability_highest_index.append(probability_higher)

    # Update strategy ('population' array) based on the lowest score.
    population_new = population.copy()

    #for agent_index in agent_indices:

    for agent_index in range(100):
        if agent_index not in agent_highest_indice:
            population_new[agent_index] = 1 if population[agent_index] == 0 else 0


    # Update 'population' for the next iteration.
    population = population_new
    scores = update_scores(population)

    # Calculate the sum of the first 'high' and last 'low' elements in 'population'.
    sum_high_agents = sum(population[:elements])
    sum_low_agents = sum(population[-(n - elements):])
    sum_population_high.append(sum_high_agents / high)
    sum_population_low.append(sum_low_agents / low)

    # Print information about the comparison in this iteration.
    print(f"Iteration {update_step + 1}:")


# Print the final population.
print("Final Population:", population)
```

B.6. Chapter 3-Multi-round-mechanism 1

We provide the code for mechanism 1, and the final outcome presented in Chapter 4.4 is the average result across 1000 iterations.

```
# Generate a population of 100 agents with random strategies (0 or 1)
population = [random.randint(0, 1) for _ in range(100)]

# Define the threshold for high-type agents (theta)
high_threshold = theta*n

# Update scores
updated_scores = update_scores(population)

score_indices = list(range(1, 101))

# Shuffle the list of agents
random.shuffle(score_indices)

# Pair agents into 50 pairs
pairs = [(score_indices[i], score_indices[i + 1]) for i in range(0, 100, 2)]
```

```
# First round of selection
selected_agents_round1 = []

for pair in pairs:
    agent1, agent2 = pair

    if updated_scores[agent1 - 1] > updated_scores[agent2 - 1]:
        selected_agents_round1.append(agent1)
    elif updated_scores[agent1 - 1] < updated_scores[agent2 - 1]:
        selected_agents_round1.append(agent2)
    else:
        selected_agents_round1.extend(pair)

# Count the number of elements under high_threshold in selected_agents_round1
high_selected_round1 = sum(1 for agent in selected_agents_round1\
if agent <= high_threshold)

# Print Scores of Selected Agents Round 1
print("Scores of Selected Agents Round 1:", [updated_scores[i - 1]\
for i in selected_agents_round1])
print("Number of elements under high_threshold in Selected Agents Round 1:",\
high_selected_round1)

# Check if there's an odd number of agents left after round 1
if len(selected_agents_round1) % 2 == 1:
    # Select one agent directly to the next round
    selected_agents_round2 = [selected_agents_round1.pop()]

    # Pair the remaining agents for comparison
    remaining_pairs = [(selected_agents_round1[i],\
    selected_agents_round1[i + 1]) for i in range(0, len(selected_agents_round1), 2)]

    # Second round of selection
    for pair in remaining_pairs:
        agent1, agent2 = pair

        if updated_scores[agent1 - 1] > updated_scores[agent2 - 1]:
            selected_agents_round2.append(agent1)
        elif updated_scores[agent1 - 1] < updated_scores[agent2 - 1]:
            selected_agents_round2.append(agent2)
        else:
            selected_agents_round2.extend(pair)
else:
    # Pair the agents for the second round
    selected_agents_round2 = []

    for pair in zip(selected_agents_round1[0::2], selected_agents_round1[1::2]):
        agent1, agent2 = pair

        if updated_scores[agent1 - 1] > updated_scores[agent2 - 1]:
            selected_agents_round2.append(agent1)
        elif updated_scores[agent1 - 1] < updated_scores[agent2 - 1]:
            selected_agents_round2.append(agent2)
        else:
            selected_agents_round2.extend(pair)
```

```
# Count the number of elements under high_threshold in selected_agents_round2
high_selected_round2 = sum(1 for agent in selected_agents_round2\
if agent <= high_threshold)


print("Selected Agents Round 1:", selected_agents_round1)
print("Number of elements under high_threshold in Selected Agents Round 1:",\
high_selected_round1)
print("Selected Agents Round 2:", selected_agents_round2)
print("Number of elements under high_threshold in Selected Agents Round 2:",\
high_selected_round2)

agents in Round 2:
selected_scores_round2 = [updated_scores[i - 1] for i in selected_agents_round2]
print("Scores of Selected Agents Round 2:", selected_scores_round2)

# Calculate proportions
proportion_high_selected_round1 = high_selected_round1 / len(selected_agents_round1)
proportion_high_selected_round2 = high_selected_round2 / len(selected_agents_round2)

print("Proportion of high_selected_round1/selected_agents_round1:",\
proportion_high_selected_round1)
print("Proportion of high_selected_round2/selected_agents_round2:",\
proportion_high_selected_round2)
```

B.7. Chapter 3-Multi-round-mechanism 2

We provide the code for mechanism 2, and the final outcome presented in Chapter 4.4 is the average result across 1000 iterations.

```
# Generate a population of 100 agents with random strategies (0 or 1)
population = [random.randint(0, 1) for _ in range(100)]

# Define the threshold for high-type agents (e.g., 50)
high_threshold =  theta*n

# Update scores
updated_scores = update_scores(population)

# Assign score indices to agents
score_indices = list(range(1, 101))

# Shuffle the list of agents
random.shuffle(score_indices)

# Divide agents into 25 groups of 4
agent_groups = [score_indices[i:i + 4] for i in range(0, 100, 4)]

selected_agents_round1 = []

for group in agent_groups:
    # Sort agents within the group based on their scores in descending order
    sorted_group = sorted(group, key=lambda agent:\
    updated_scores[agent - 1], reverse=True)

    # Select agents with the highest score in the sorted group
```

```
    max_score = updated_scores[sorted_group[0] - 1]
    selected_agents_round1.extend(agent for agent in sorted_group\
    if updated_scores[agent - 1] == max_score)

# Print Scores of Selected Agents Round 1
print("Scores of Selected Agents Round 1:", [updated_scores[i - 1]\
for i in selected_agents_round1])

# Count the number of elements under high_threshold in selected_agents_round1
high_selected_round1 = sum(1 for agent in selected_agents_round1\
if agent <= high_threshold)

print("Selected Agents Round 1:", selected_agents_round1)
print("Number of elements under high_threshold in Selected Agents Round 1:",\
high_selected_round1)

# Calculate proportions
proportion_high_selected_round1 = high_selected_round1 / len(selected_agents_round1)
print("Proportion of high_selected_round1/selected_agents_round1:",\
proportion_high_selected_round1)
```

Bibliography

Boydell, J., Deutsch, B., & Remillard, B. (2005). *You're not the person i hired!: A ceo's survival guide to hiring top talent*. AuthorHouse. https://books.google.nl/books?id=k7WjXbAs9P8C

Brown, G. W. (1951). Iterative solution of games by fictitious play. In T. C. Koopmans (Ed.), *Activity analysis of production and allocation*. Wiley.

Ferguson, T. (2020). *A course in game theory*. World Scientific Publishing Company. https://books.google.nl/books?id=knr1DwAAQBAJ

Gadagkar, R. (2005). The logic of animal conflict. *Resonance, 10*, 5–5. https://doi.org/10.1007/BF02837640

Hvide, H. K., & Kristiansen, E. G. (2003). Risk taking in selection contests. *Games and Economic Behavior, 42*(1), 172–179.

Karlin, A., & Peres, Y. (2017). *Game theory, alive*. American Mathematical Society. https://books.google.nl/books?id=9_vFDgAAQBAJ